LIBRARY
RESEARCH REPORTS DIVISION
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93940

NPS-54-86-014

NAVAL POSTGRADUATE SCHOOL

Monterey, California

A MODEL MANAGEMENT SYSTEM FOR COMBAT SIMULATION

by

Daniel R. Dolk

November 1986

Approved for public release; distribution unlimited.

Prepared for:
U. S. Army TRADOC Systems Analysis Activity
White Sands Missile Range, NM 88002-5502

NAVAL POSTGRADUATE SCHOOL
Monterey, California

RADM. R. C. Austin
Superintendent

David A. Schrady
Provost

The research summarized herein was accomplished with resources provided by the U. S. Army TRADOC Systems Analysis Activity.

This report was prepared by:

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) NPS54-86-014	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION val Postgraduate School 6b OFFICE SYMBOL (If applicable) Code 54	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and ZIP Code) de 54Dk nterey, CA 93943	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING / SPONSORING ORGANIZATION S. Army TRADOC Systems Analysis Activity 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code) rector TN: ATOR-TCA ite Sands Missile Range, NM 88002-5502	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: N62271 PROJECT NO: 86 TASK NO.: M WORK UNIT ACCESSION NO: 0204

11 TITLE (Include Security Classification)
A Model Management System for Combat Simulation

12 PERSONAL AUTHOR(S)
Daniel R. Dolk

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
	FROM 10/85 TO 09/86	1986 November 12	71

16 SUPPLEMENTARY NOTATION

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Model management, information resource dictionary system, structured modeling, Jackson system development, discrete event simulation, combat simulation.

19 ABSTRACT (Continue on reverse if necessary and identify by block number)
e design and implementation of a model management system to support combat deling is discussed. Structured modeling is introduced as a formalism for presenting mathematical models. A relational information resource dictionary stem is developed which can accommodate structured models. An implementation described. Structured modeling is then compared to Jackson System Develop-at (JSD) as a methodology for facilitating discrete event simulation. JSD currently better at representing the dynamic aspects of simulation whereas ructured modeling excels in representing the static aspects. A structured del of an existing combat model is presented. Finally, recommendations are le to strengthen structured modeling as a tool for discrete event simulation.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	

22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code)	22c OFFICE SYMBOL
Daniel R. Dolk	408-646-2260	

FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

A MODEL MANAGEMENT SYSTEM FOR COMBAT SIMULATION

by

Daniel R. Dolk

November 1986

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This report describes the results of a project which the Naval Postgraduate School has been conducting for TRASANA during the fiscal year 1985-86. The project originated from a 1984 Masters's thesis [Napoliello and Stone 1984] which examined a prototype version of the PDF (Programmer's Design Facility) program for doing Jackson Structured Programming. The objective was to determine whether PDF was a sufficiently powerful tool to serve as a means for facilitating Jackson System Methodology as the standard software development environment at TRASANA. At a meeting held at Naval Postgraduate School in the summer of 1984 and attended by representatives of TRASANA, Fort Leavenworth, Fort Lee, and Naval Postgraduate School, it was agreed that a more powerful tool than PDF was required for this purpose. This project is an attempt to define and develop such a tool.

1.1 Objectives

PDF was found lacking in several critical areas:

1. machine-dependent
2. no data abstraction or data description facilities
3. no support for Jackson System Development
4. limited and unwieldy "automatic" code generation

The primary objective of the project is to build a software environment which supports the full Jackson software development methodology (JSD and JSP), or its equivalent, as applied to TRASANA's mission of designing and implementing combat simulation models.

Desirable features of this tool include a dictionary capability, a database management system (DBMS) implementation, graphical interfaces, and a reasonable migration path from desktop to mainframe systems. The prototype system is to be implemented on a desktop computer compatible with TRASANA specifications.

1.2 Methodology

At the time this project was being considered, Prof. A. M. Geoffrion from UCLA was developing an integrated approach to modeling known as **structured modeling (SM)** [Geoffrion 1985]. Because of the flexibility of SM and its potential benefits for modeling in general, it was decided to build a system based on SM and to embed the Jackson-oriented tools within this environment. The motivation for this approach was that such a system would provide powerful **model management** facilities which would significantly complement the benefits of the Jackson methodology.

Adopting SM as the lingua franca raises several key issues:

1. What is the architecture for a model management system (MMS) based on SM?

2. Is SM an appropriate medium for discrete event simulation modeling?

3. Is SM an appropriate medium for the Jackson approach to information system development?

The work on this project attempts to answer these questions.

1.3 Structure of Report

The remainder of this report is structured as follows. Section 2 provides a brief survey of structured modeling. Section 3 describes the current state of the prototype MMS focusing on implementation of an information resource dictionary (IRDS). Section 4 analyzes the extent to which SM and JSD/JSP are compatible modeling approaches. Section 5 presents a preliminary SM representation of the ONEC combat simulation model and discusses the problems encountered in deriving this representation. Section 6 summarizes the findings of the project to date and suggests a strategy for continuing the project.

1.4 External References

There are several references which explain aspects of this project in much more detail than this report. Geoffrion's work on SM [Geoffrion 1985, 1986a, 1986b] provides a full formal treatment, an introductory tutorial, and a comparison with other modeling approaches respectively. The development of a relational IRDS and its extension to SM are presented in [Dolk and Kirsch 1986, Dolk 1986]. Investigation of the relationships between JSD and SM and JSP and SM were subcontracted to Professors Jeffrey Kottemann and Jack Stott of the University of Hawaii respectively. Their findings are presented in [Kottemann 1986] and [Stott 1986] and are incorporated in this report where described in the appropriate sections.

2. STRUCTURED MODELING

Structured modeling has been developed by A. M. Geoffrion to facilitate and integrate as many aspects of the modeling process as possible.

> "Structured modeling endeavors to provide a formal mathematical framework, language, and computer-based environment for conceiving, representing, and manipulating a wide variety of models." [Geoffrion 1986a]

Although structured modeling was primarily motivated by problems encountered in the management science/operations research (MS/OR) community, much cross-fertilization has occurred with other model-oriented disciplines such as database systems, software engineering, and artificial intelligence. As a result structured modeling has evolved into a very general approach with potential application to many different fields. This versatility qualifies it as a powerful, integrative tool in support of information resource management.

We provide here just enough material about the objectives and basics of structured modeling to motivate its application to information resource management. The reader is encouraged to consult Geoffrion [1985, 1986a, 1986b] for a full, comprehensive treatment.

2.1 Objectives of Structured Modeling

The motivation for structured modeling arises from a variety of considerations:

1. Low productivity and acceptance of MS/OR: This has been a long-standing problem which has its roots in cumbersome user interfaces supplied by modeling software and a lack of communication between management and technically oriented MS/OR practitioners.

2. Popularity of personal computers and spreadsheets: Spreadsheets have proven that people will build and use models given the right tools. Incorporating this "informal" modeling activity into organizationally productive channels requires an integrated approach to modeling.

3. Maturing database technology: The recent, increased availability of relational database systems significantly facilitates user access to data resources. Modeling systems should take advantage of this capability instead of relying on outdated file processing methods. This is particularly relevant for information resource management which is firmly based on database management principles.

A modeling system which successfully addresses these considerations must have at least the following characteristics:

1. A conceptual framework which defines a single model representation;

2. Independence of the model representation from both model solution operators and underlying data associated with specific model instances;

3. The ability to capture a wide range of MS/OR mathematical models as well as other conceptual models related to the disciplines of database design and software engineering;

4. Support for the overall modeling life cycle;

5. Full use of data management facilities as embodied in database management systems;

6. Personal computer implementation and spreadsheet compatibility in the form of immediate expression evaluation.

Structured modeling is a very general approach to the problems and activities associated with modeling. Previous modeling systems have tended to be application specific (e.g.: linear programming systems) and have subsequently failed to satisfy one or more of the above requirements (see [Geoffrion 1986b] for a more complete review). Although the objectives of structured modeling are ambitious, they are nevertheless consistent with, and necessary for, the successful implementation of model management.

2.2 Fundamentals of Structured Modeling

Structured modeling is a unified modeling framework based on acyclic, attributed graphs. There are three basic structures which comprise this framework: **elemental**, **generic**, and **modular**.

Models are defined in terms of elements which may be partitioned into genera (pl. of "genus") and further aggregated into modules. There are five element types: **primitive entity**, **compund entity**, attribute (plus a variation called a **variable attribute**), **function**, and **test**. Primitive entities are existential in nature and have no value mathematically. Compound entities reference other entities already defined and require no value. Attributes associate a certain property and value with an entity or combination of entities. Variable attributes are like attributes except that values may not be specified. Variable attributes most resemble decision variables in a linear programming model. Function elements associate a rule and value with an entity or combination of entities. Function elements resemble mathematical equations. Test elements are like function elements with a boolean (True,False) value. Test elements constraints in mathematical programmming models.

Each element has a calling sequence which identifies other elements directly referenced. The calling sequence captures the

cross-references among model elements and can be derived directly from the graphical representation. The **elemental structure** of a model is a nonempty, closed, finite, acyclic collection of elements. Acyclicity implies that there is no sequence of calling sequences which turns out to be "circular".

The **generic structure** of a model is a partitioning of the elemental structure such that there is one partition (genus) for each element type. Genus is similar to the notion of set or class. Partitioning must satisfy **generic similarity** in that every element in a genus must have the same number of calling sequence segments and all elements in a given calling sequence segment must belong to the same genus. Partitioning enforces strong typing in that a single element may belong to one, and only one, genus.

Modular structure is a tree defined on the generic structure all of whose leaves are genera and all of whose non-terminal nodes are modules. Modular structure allows genera to be grouped in ways that might be conceptually meaningful to users. It facilitates a view mechanism which allows users to view the model at different levels of abstraction. Not all modular structures are permitted, however. Only those which satisfy **monotone ordering**, i.e. those which admit an indented list representation with no forward references (genera which call genera further down the list), are allowed.

A **structured model** consists of an elemental stucture, a generic structure which satisfies generic similarity, and a modular structure with monotone ordering.

2.3 Example: The Transportation Problem

The easiest way to absorb the terminology is by examining a simple example. The transportation model is a familiar model discussed in all introductory texts on management science. The scenario entails plants which produce a single product for shipment to customers. Every plant has a maximum supply capacity and every customer has an exact demand requirement. For every link which exists between a plant and a customer, there is an associated unit transportation cost. The model allows us to evaluate various transportation flows over the links which satisfy production capacities and demand requirements in terms of the resultant total transportation cost.

Primitive entities include every instance of a plant (assume plants in Dallas and Chicago) and every instance of a customer (assume customers in Seattle, Boston, and Atlanta). Compound entities include every link between a plant and a customer (assume links Dallas-Seattle, Dallas-Atlanta, Dallas-Boston, Chicago-Seattle, and Chicago-Boston). Attributes include the supply capacity for each plant, the demand requirement for each customer, and the transportation cost for each link. The flow for each link is a variable attribute. Test elements consist of the supply constraint for each plant and the demand constraint

for each customer. A single function element describes the total transportation cost.

Calling sequences for the transportation model capture the functional dependencies among the model elements. In general, attributes of an entity will have that entity in its calling sequence. Thus each supply capacity has a plant in its calling sequence, each demand requirement a customer, and each transportation flow and cost a link. Compound entities have the entities which they depend upon in their calling sequence thus each link has both a customer and a plant in its calling sequence. The supply and demand constraints depend upon supply and demand respectively as well as the variable attribute transportation flow. Finally, the total cost function element depends upon the cost and flow.

The elemental structure of this model captures all the associations among the elements just described and can be represented as an acyclic graph. In general, this graph will be too detailed to be of much use. We can partition the elemental structure into a generic structure by defining the following genera: primitive entity (PLANT and CUSTOMER), compound entity (LINK), attribute (SUPPLY, DEMAND, COST), variable attribute (FLOW), test (T:SUP and T:DEM), and function (TOTAL_COST). The similarity requirement insures that calling sequences for the genera are exactly as those for the elements. The graphical representation of the generic structure is more concise and meaningful (Figure 2-1).

A modular structure can be imposed upon the generic structure to represent higher levels of abstraction by grouping related genera. For example, we may want to aggregate segments of the overall model into sales, production, and distribution components. This can be done by defining modules &SALES, &PROD, and &DIST (the "&" by convention refers to a module) which are rooted trees whose leaves are genera (Figure 2-2). The modular structure can, in a sense, be viewed as orthogonal to the generic structure while preserving the acyclic nature of the latter. Modular structure facilitates different views of the model corresponding to the level of detail which a user desires. The monotone ordering of the modular structure insures that these different views all preserve the acyclicity of their corresponding generic structures.

The modular structure can be represented as a modular outline where a preorder traversal of the hierarchy is performed and levels of indentation correspond to levels of the hierarchy indentation (Figure 2-2). The monotone requirement for modular structures insures that there are no forward references in this outline, i.e. no genus in the outline has any genera in its calling sequence which appear later in the outline. This outline is fleshed out into a model schema by adding relevant information such as genus type, calling sequence, mathematical representation, and natural language interpretation.

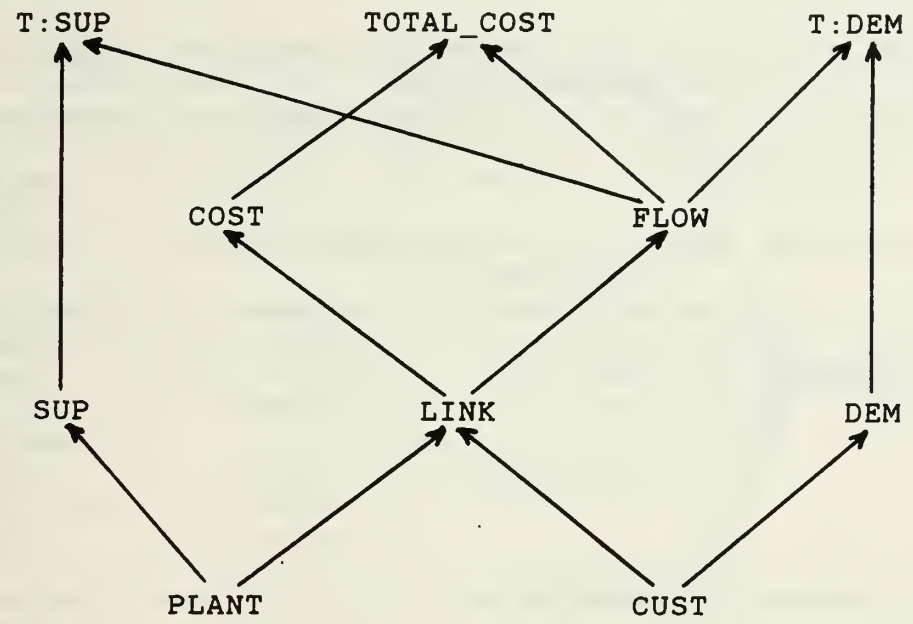


Figure 2-1: Generic Structure of Transportation Model
(from [Geoffrion 1985])

2.4 Structured Modeling and Model Management

Structured modeling provides a robust medium for model representation which, depending upon implementation strategy, satisfies all of the objectives stated in Section 2.1. Structured modeling clearly offers a substantial framework upon which model management can be built. What is lacking, however, is the proper organizational information resource perspective which is central to the notion of model management. In particular, the perception of models as an organizational resource which can be shared amongst many users is critical.

Structured modeling focuses on the individual user and emphasizes personal, or desktop, computer implementation. This is obviously an appealing prospect given the versatility and availability of PC hardware/software tools. One cannot afford to ignore the organizational ramifications of this approach, however. Within the information resource environment, there is always the tradeoff between management's desire for control and end users' desire for more effective computing. Spreadsheets offer the ideal example. Although spreadsheets have provided a very effective individual decision making tool, examples abound of organizational problems resulting from incompatible software and/or hardware, incorrect modeling practices which lead to bad decisions, conflicting models and data, and other phenomena

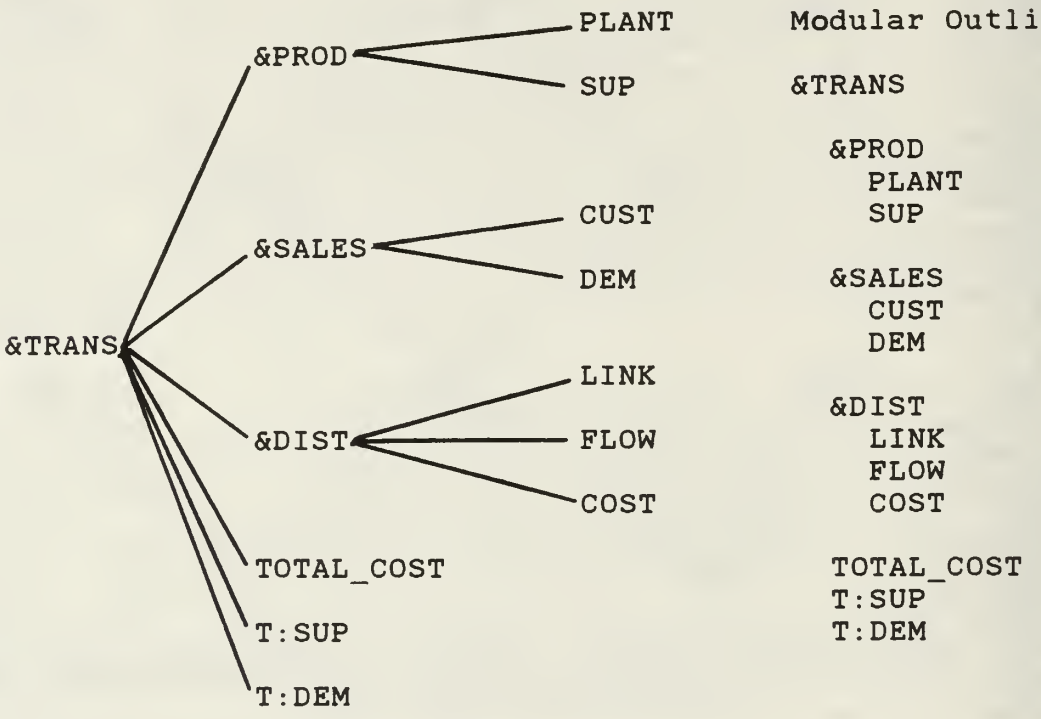


Figure 2-2: Modular Structure and Outline for Transportation Model (from [Geoffrion 1985])

related to uncontrolled usage. Organizations are still searching for ways to control this situation.

The recognition of data as a resource and the need for sharing and centralized control of data earmarks the data management approach. The same is true for models and model management. Model management is a part of organizational information resource policy governing the control and sharing of modeling resources. What this means is that we must examine structured modeling in the context of information resource management. This requires extending structured modeling to a multi-user, centralized, mainframe environment.

The primary control element in an IRM environment is the IRDS. This database contains descriptions of, and relationships among, other information resources. The IRDS is ideally instrumental in the planning, administration, and operation of an organization's information processing activity. A natural migration path, therefore, for incorporating model management into IRM is to implant structured modeling into an existing IRDS. The next section shows how this can be done with only slight modifications to a Federal standard IRDS.

3. AN ORACLE-BASED MMS PROTOTYPE

This section describes the underlying principles and current stage of development for the model management system prototype. The discussion contained herein is distilled from previous IRDS work [Dolk and Kirsch 1986] and a current paper on the application of IRDS to SM representation [Dolk 1986].

3.1 A Relational IRDS (RIRDS) Based on Federal Standards

The information resource dictionary system (IRDS) is the basic tool which will support all TRASANA model and software development in this project. Once the IRDS is built, then the model management system and/or JSD/JSP will be added to this foundation.

3.1.1 Dictionary concepts

Many different terms are used as synonyms for dictionaries but the one most commonly applied is **information resource dictionary system (IRDS)**. An IRDS is essentially a knowledge base about an organization's information resources. It includes capabilities for describing and storing data about information resources as well as retrieving and manipulating this data. Since the data in an IRDS describes other data, it is often referred to as **metadata** and the administration of the dictionary is correspondingly termed **metadata management**.

Dictionary systems typically have two distinct components: the **dictionary** and the **directory**. The dictionary aspect describes **what** information resources exist, what they mean, what their structures are, and how they interrelate. The directory, on the other hand, describes **where** these resources are located and how they are accessed.

Dictionaries are classified as either **passive** or **active**. A passive system does not interact dynamically with any other operational system, i.e. no system depends on the dictionary for its metadata. An active dictionary, on the other hand, generates metadata for one or more processes and is the sole source of that metadata. A database management system (DBMS), for example, may use an active dictionary for all information concerning the description of data items in operational databases. Passive dictionaries are used essentially for documentation and must be updated independently from the operational environment they describe. Active systems are more powerful in implementing control mechanisms but require more overhead in interfacing with other systems. A common implementation strategy is to build a passive system first and then extend it to an active one for selected applications. Techniques for doing this in the DCSPLANS environment will be discussed later in this section.

IRDS are also characterized as either **DBMS-dependent** or **free-standing (DBMS-independent)**. A DBMS-dependent IRDS needs an underlying DBMS to perform metadata retrieval and manipulation. A freestanding IRDS supplies all those functions internally. A DBMS-dependent IRDS can be be built from scratch more readily but is constrained to operate in an environment containing the underlying DBMS. A free-standing IRDS is more versatile in this regard yet more costly to build.

In summary, the active/passive designation refers to whether or not other operational systems need the IRDS for their metadata whereas the DBMS-dependent/independent classification refers to whether or not the IRDS needs a DBMS to perform its manipulation functions. The usual approach is to first build a passive, DBMS-dependent IRDS and then convert it to an active, dependent system. Details for accomplishing this are presented in the remainder of this section.

3.1.2 FIPS IRDS

It has been estimated that the Federal government can realize $120 million in benefits by the early 1990's from the use of a standard IRDS. As a result, the National Bureau of Standards has developed specifications for an IRDS which will form the basis for a Federal Information Processing Standard (FIPS) IRDS. These specifications include many of the functions available in existing commercial dictionary systems while also providing flexibility for tailoring the IRDS to specific information administration requirements.

The central feature of the FIPS IRDS is the core **system-standard** schema (core) which describes the logical structure of the IRDS itself. The core consists of entity, attribute, and relationship types as shown in Appendix A. Entity types correspond to the various objects which exist in an IRM environment such as files, programs, and users. Attribute types are simply descriptors of entity types. The core supports three distinct name attributes for each entity: ACCESS-NAME, DESCRIPTIVE-NAME, and ALTERNATE-NAME. ACCESS-NAME is a short, easy to use, and unique name with which the user will most frequently interact whereas DESCRIPTIVE-NAME provides a more meaningful, but also unique, name. ALTERNATE-NAME allows multiple aliases to be associated with any one entity. Relationship types capture the important associations between entities that exist in an information resource environment. An important feature of the FIPS IRDS core is that all relationships between entity types are binary in nature. Further, there are constraints as to which entity types are allowed to participate in which relationships (see Table A-4 in Appendix A). For example **PROCESSES(system,file)** is legal but not **PROCESSES(file,system)** since a file cannot process a system.

The intent of the FIPS IRDS specifications is that the core serve as a common baseline from which to implement IRM. It's not expected that the core will be sufficiently robust to support all IRM environments, however. As a result, the core is

characterized as being **extensible** in that additional entity, attribute, and relationship types can be added to support unique requirements. Thus, each installation has the flexibility to tailor the IRDS to their specific information resource environment.

3.1.3 A relational model of FIPS IRDS (RIRDS)

The FIPS specifications mandate that an IRDS implementation will be considered in compliance if it supports fully the core and additionally supports either a panel-driven (menu-driven) or command language interface. A DBMS-dependent IRDS is a logical way to incorporate a command language interface since DBMS's automatically provide data sublanguages with which to describe, manipulate, and control data. In particular, a relational DBMS (RDBMS) based on the SQL data sublanguage provides an ideal environment for implementing the FIPS IRDS. This section describes such an implementation performed with the relational ORACLE DBMS developed by the ORACLE Corporation. No claims are made for the relative superiority of ORACLE vis-a-vis other systems. In fact, the implementation described herein should be easily transportable to other DBMS environments. Familiarity with SQL is assumed in the following discussion. See Appendix B for a concise summary of SQL.

A simple relational model of the core types is shown in Figure 3-1. Notice that the entity types have many attibutes in common but that some attributes (e.g.: lines-code) are only associated with a subset of the entity types as shown in Table A-2. This relational version of the core can be simplified into 2 relations, ENTITY and RELSHIP (Figure 3-2a) by using the relational view mechanism as embodied in the SQL language. For example, we can impose the view PROGRAM (as shown in Figure 3-1) on ENTITY with the following SQL command:

```
CREATE VIEW PROGRAM AS
(SELECT ANAME,DNAME,ADDED_BY,DATE_ADDED,MOD_BY,LAST_MOD,
        NMODS,DUR_VALUE,DUR_TYPE,LINES_CODE,COMMENTS,SECURITY
 FROM   ENTITY
 WHERE  ETYPE='PROGRAM');
```

Similarly, we can impose the view PROCESSES (as shown in Figure 3-1) on RELSHIP:

```
CREATE VIEW PROCESSES AS
(SELECT E1NAME, E1TYPE, E2NAME, E2TYPE
 FROM   RELSHIP
 WHERE  RTYPE='PROCESSES');
```

Notice that the attribute **etype** in ENTITY must be one of the entity types shown in Figure 3-1 and **rtype** in RELSHIP must be one of the relationship types. By creating a view for each entity type and relationship type in the FIPS IRDS, we create the logical equivalent of Figure 3-1 using only two underlying relations.

Entities and Attributes

SYSTEM(aname,dname,added-by,date-added,mod-by,last-mod,
nmods,dur-value,dur-type,comments,descr,security)

PROGRAM(aname,dname,added-by,date-added,mod-by,last-mod,
nmods,dur-value,dur-type,lang,lines-code,comments,
descr,security)

MODULE(aname,dname,added-by,date-added,mod-by,last-mod,
nmods,dur-value,dur-type,lines-code,comments,descr,
security)

FILE(aname,dname,added-by,date-added,mod-by,last-mod,nmods,
nrecs,comments,descr,security)

RECORD(aname,dname,added-by,date-added,mod-by,last-mod,nmods,
rec-cat,comments,descr,security)

ELEMENT(aname,dname,added-by,date-added,mod-by,last-mod,
nmods,data-class,low-range,high-range,comments,
descr,security)

DOCUMENT(aname,dname,added-by,date-added,mod-by,last-mod,
nmods,doc-cat,comments,descr,security)

USER(aname,dname,added-by,date-added,mod-by,last-mod,
nmods,comments,descr,location,security)

Relationships

All relationships have the same attributes and keys:

REL(e1name,e1type,e2name,e2type)

where e1name,e2name are the entity instances

e1type,e2type are the entity-types of which e1name,
e2name are instances, respectively

REL is any of the relationships CONTAINS, PROCESSES,
RUNS, RESP_FOR, CALLS, GOES_TO, DERIVED_FROM,
ALIAS and KWIC.

Integrity Constraints

(Please see Table A-4.)

Figure 3-1: Relational IRDS (RIRDS)

ENTITY(ename,etype,dname,added-by,date-added,mod-by,
last-mod,nmods,dur-value,dur-type,comments,descr,
security,lang,lines-code,nrecs,rec-cat,data-class,
doc-cat)

RELSHIP(rtype,e1name,e1type,e2name,e2type,access-method,
frequency,rel_pos)

(a) Simplified relational representation of the IRDS core entity-relationship model

Meta-Entities, -Attributes, -Relationships

ENT_TYPE(aname,dname,added-by,date-added,mod-by,last-mod,
nmods,comments,descr,security)

ATT_TYPE(aname,dname,added-by,date-added,mod-by,last-mod,
nmods,comments,descr,security)

REL_TYPE(aname,dname,added-by,date-added,mod-by,last-mod,
nmods,comments,descr,security)

(b) RIRDS schema description

Figure 3-2: Simplified Relational IRDS Model

3.1.4 Self-descriptive IRDS

Since an IRDS describes information resources, it should be able to describe itself. This implies that the information resource administrator should be able to determine from the IRDS which entity, attribute, and relationship types the IRDS supports as well as the relationship constraints in effect (see Table A-4). Self-descriptive capabilities facilitate a strong integrity checking mechanism as we show later. The relational model described so far has very limited self-descriptive features. Although we could determine which entity and relationship types exist by the following two commands:

```
SELECT UNIQUE ETYPE FROM ENTITY;

SELECT UNIQUE RTYPE FROM RELSHIP;
```

this is far from ideal. For one thing, it's possible that the

IRDS may support an entity type (e.g.: MODULE) for which no instances have been entered yet. In this case, the first query would not show tha MODULE was an entity type. A similar situation holds for relationship types in the second query. Further, the relational model in its current form has no way of describing the relationship constraints.

In order to make the IRDS self-descriptive, three new relations must be added corresponding to each of the types: ENT_TYPE, ATT_TYPE, and REL_TYPE (Figure 3-2b). These meta-relations describe relations or views existing at the ENTITY/RELSHIP level. For example, the domain of **ENTITY.etype** is defined by the set of values of **ENT_TYPE.aname**; similarly for **RELSHIP.rtype** and **REL_TYPE.aname**. Now the entity and relationship types can be listed independent of whether actual instances of these types are in the database:

```
SELECT ANAME FROM ENT_TYPE;

SELECT ANAME FROM REL_TYPE;
```

Further, the relationship constraints can now be represented explicitly in the IRDS as instances in the appropriate relationship view. For example to represent the constraint PROCESSES(system,file) requires an entry in RELSHIP as follows:

```
RELSHIP('processes', 'system', 'ent_type', 'file', 'ent_type')
```

Once the constraints have been entered, the administrator can retrieve information about the IRDS itself. For example, to determine which relationships the entity type PROGRAM can legally participate in, the administrator would issue the following SQL command:

```
SELECT RTYPE, E1NAME, E2NAME FROM RELSHIP
WHERE E1NAME='program' OR E2NAME='program'
```

The RIRDS deviates from the FIPS IRDS core in two respects: physical representation and multiple attributes. The entity types BIT-STRING, CHARACTER-STRING, FIXED-POINT, and FLOAT have been omitted as well as the relationship type REPRESENTED-AS. These types are concerned with the physical representation of data (**ELEMENT** entities) whereas the rest of the core is concerned with logical relationships. Further the FIPS IRDS approach in this case precludes many realistic situations wherein an element entity (e.g., SOCIAL_SECURITY_NUMBER) may appear as a FIXED-POINT in one file and CHARACTER-STRING in another file. We recommend inbstead embedding this information as an attribute type (e.g., FORMAT) in the FILE-CONTAINS-ELEMENT relationship type.

Multiple attribute types have been omitted from the RIRDS since these violate first normal form. Multiple attributes are those which may have more than one occurrence for each entity. For example, the attribute type LOCATION may have several values for a file which is distributed at various nodes in a network.

Multiple attributes require new relations to be defined in addition to the two shown in Figure 3-2a and thus complicate the model. In the cases where these attributes are vital (e.g.: ALTERNATE-NAME and CLASSIFICATION), new relationship types have been defined (ALIAS and KWIC, respectively) to accommodate the situation. ALIAS provides a valuable synonym capability which allows multiple names to be assigned to the same entity. It's defined by the following SQL command:

```
CREATE VIEW ALIAS AS
(SELECT E1NAME, E1TYPE, E2NAME FROM RELSHIP
 WHERE RTYPE = 'alias')
```

The key-word-in-context (KWIC) allows entities to be classified according to user-chosen categories and facilitates queries of the kind, "list all entities associated with REENLISTMENT". It's defined analogously to ALIAS:

```
CREATE VIEW KWIC AS
(SELECT E1NAME, E1TYPE, E2NAME FROM RELSHIP
 WHERE RTYPE = 'kwic')
```

Other multiple attributes which the data administrator considers vital can be included using the same kind of strategy.

3.1.5 Data integrity

Data integrity is one of the critical problems in any information processing. The IRDS affords a built-in consistency checking mechanism for bringing this problem under control.

The self-descriptive capability of the dictionary allows the IRDS to check whether its contents are consistent with its own logical description. For example, someone may inadvertently have entered the information PROCESSES('pos_edit', 'file', 'pos_entry', 'program') which violates the acceptable constraints for PROCESSES as shown in Table A-4 since a file cannot process a program. The following SQL command identifies all violations of PROCESSES constraints ("!" is equivalent to "NOT"):

```
SELECT * FROM PROCESSES
WHERE E1TYPE != 'ent_type' AND E2TYPE != 'ent_type' AND
     (E1TYPE, E2TYPE) NOT IN
      (SELECT E1NAME, E2NAME FROM PROCESSES
       WHERE E1TYPE = 'ent_type' AND E2TYPE = 'ent_type')
```

The way this query works is that the subquery (2nd SELECT clause) retrieves the set of all pairs of entity-types which may legally participate in PROCESSES. The first SELECT clause then identifies and displays any pairs of entities in the meta-data appearing in PROCESSES whose entity types do not fall in that set. All invalid occurrences can subsequently be deleted from the database by simply changing "SELECT *" to "DELETE" in the above query.

Notice that the above query can be used for any relationship type by simply changing "PROCESSES" to the appropriate relationship type name. A global consistency check can be performed by replacing "PROCESSES" with "RELSHIP". This would check all constraints in Table A-4 for possible violations, thus one SQL command is all that's required to determine the integrity of the data in the IRDS itself.

The power of SQL in integrity checking is somewhat offset by the complexity of the required commands. This can be neatly circumvented, however, by taking advantage of macro facilities which most relational DBMS provide. In the ORACLE system, for example, the global consistency query might be saved as the macro GLOBAL_CHECK which could then be invoked directly without having to know the complexities of SQL. One of the duties of the data administrator is to define an appropriate set of such macros for the user commmunity (see Appendix C for a representative sample).

3.2 Extending the IRDS to Capture Structured Modeling

In the same way that a DBMS provides a tool for implementing data management, a model management system (MMS) is required to support model management. An MMS must provide model description, manipulation, and control functions and must support the sharing of models and their underlying data. A preliminary version of an MMS based on structured modeling can be implemented by a simple extension of the IRDS described above.

In order to accommodate the representation of structured models, the IRDS core must be extended to include new entity types corresponding to the genus types (primitive entity (pe), compound entity (ce), attribute (att), variable attribute (va), test (test), and function (fcn)) in structured modeling (Figure 3-3). The entity type **genus** is also a useful, although not strictly necessary, addition. The entity type **model** should also be added to reflect models as an important resource. These entity types will then be implemented as entities by establishing the appropriate views on the ENTITY relation. Constraints governing generic structure (i.e., acceptable calling sequences) and modular structure must also be defined in the IRDS.

With this extended IRDS core, it is now possible to represent structured models in relational form. The IRDS representation of the transportation model used as an example in Section 2 is shown in Figure 3-4.

This representation facilitates several different kinds of queries. Appendix C enumerates some model validity commands which could be established by the data/model administrator. Calling sequences for a particular genus (e.g.: FLOW) can be determined via the following SQL command:

```
SELECT E2NAME, E2TYPE FROM CALLS
WHERE E1NAME = 'flow'
```

Entity-Types

```
ENT_TYPE('pe', 'primitive_entity', ....)
ENT_TYPE('ce', 'compound_entity', ....)
ENT_TYPE('att', 'attribute', ....)
ENT_TYPE('va', 'variable_attribute', ....)
ENT_TYPE('test', 'test_entity', ....)
ENT_TYPE('fcn', 'function_entity', ....)
ENT_TYPE('model', 'model', ....)
```

Entities

```
PE(aname, dname, .... , doc_cat, index,
   index_stmt, gen_range, gen_rule)

CE(aname, dname, .... , doc_cat, index,
   index_stmt, gen_range, gen_rule)

ATT(aname, dname, .... , doc_cat, index,
   index_stmt, gen_range, gen_rule)

VA(aname, dname, .... , doc_cat, index,
   index_stmt, gen_range, gen_rule)

TEST(aname, dname, .... , doc_cat, index,
     index_stmt, gen_range, gen_rule)

FCN(aname, dname, .... , doc_cat, index,
    index_stmt, gen_range, gen_rule)

MODEL(aname, dname, .... , doc_cat, index,
    index_stmt, gen_range, gen_rule)
```

Integrity Constraints

```
CALLS(ce,pe)        CALLS(va,pe)        CALLS(test,test)
CALLS(att,pe)       CALLS(va,ce)        CALLS(test,fcn)
CALLS(att,ce)       CALLS(test,va)      CALLS(fcn,fcn)
CALLS(test,att)     CALLS(fcn,va)       CALLS(fcn,test)
CALLS(fcn,att)

CONTAINS(module,module)          CONTAINS(module,test)
CONTAINS(module,pe)              CONTAINS(module,fcn)
CONTAINS(module,ce)              CONTAINS(model,module)
```

Figure 3-3: IRDS Representation of Structured Modeling

Entities

```
MODEL('TRANSP', 'Transportation_Model', ....)
MODULE('&PROD', 'Source_Data', ....)
MODULE('&SALES', 'Customer_Data', ....)
MODULE('&DIST', 'Transportation_Data', ....)
PE('PLANT', 'Mfg_Plant', ....)
PE('CUST', 'Customer', ....)
CE('LINK', 'Link_Plant_&_Customer', ....)
ATT_TYPE('SUP', 'Plant_Supply_Capacity', ....)
ATT_TYPE('DEM', 'Customer_Demand', ....)
ATT_TYPE('FLOW', 'Transportation_Flow', ....)
ATT_TYPE('COST', 'Transportation_Cost', ....)
TEST('T:SUP', 'SUMj(FLOWij) <= SUPi', ....)
TEST('T:DEM', 'SUMi(FLOWij) = DEMj', ....)
FCN('TOTAL_COST', 'SUMij(COSTij * FLOWij)', ....)
```

Generic Structure

```
CALLS('SUP', 'ATT_TYPE', 'PLANT', 'PE')
CALLS('T:SUP', 'TEST', 'SUP', 'ATT_TYPE')
CALLS('DEM', 'ATT_TYPE', 'CUST', 'PE')
CALLS('T:DEM', 'TEST', 'DEM', 'ATT_TYPE')
CALLS('LINK', 'CE', 'PLANT', 'PE')
CALLS('LINK', 'CE', 'CUST', 'PE')
CALLS('COST', 'ATT_TYPE', 'LINK', 'CE')
CALLS('FLOW', 'ATT_TYPE', 'LINK', 'CE')
CALLS('T:SUP', 'TEST', 'FLOW', 'ATT_TYPE')
CALLS('T:DEM', 'TEST', 'FLOW', 'ATT_TYPE')
CALLS('TOTAL_COST', 'FCN', 'FLOW', 'ATT_TYPE')
CALLS('TOTAL_COST', 'FCN', 'COST', 'ATT_TYPE')
```

Modular Structure

```
CONTAINS('TRANSP','MODEL','&PROD','MODULE')
CONTAINS('TRANSP','MODEL','&SALES','MODULE')
CONTAINS('TRANSP','MODEL','&DIST','MODULE')
CONTAINS('TRANSP','MODEL','TOTAL_COST','FCN')
CONTAINS('TRANSP','MODEL','T:SUP','TEST')
CONTAINS('TRANSP','MODEL','T:DEM','TEST')
CONTAINS('&PROD','MODULE','PLANT','PE')
CONTAINS('&PROD','MODULE','SUP','ATT_TYPE')
CONTAINS('&SALES','MODULE','CUST','PE')
CONTAINS('&SALES','MODULE','DEM','ATT_TYPE')
CONTAINS('&DIST','MODULE','LINK','CE')
CONTAINS('&DIST','MODULE','COST','ATT_TYPE')
CONTAINS('&DIST','MODULE','FLOW','ATT_TYPE')
```

Figure 3-4: IRDS Representation of Transportation Model

A natural language summary of the transportation model for managers and a mathematical summary for modelers can be generated by the following commands respectively:

```
SELECT ENAME, ETYPE, INDEX, COMMENTS
FROM GENUS

SELECT ENAME, ETYPE, INDEX, INDEX_STMT
FROM GENUS
```

Numerous other retrievals can be made to support either modeling or administration functions.

One of the powerful features of structured modeling is that the model schema automatically defines a relational form for the underlying data (or elemental detail) associated with the model. The generic structure defines functional dependencies between model components and these dependencies can be transformed automatically into a set of relations (see [Geoffrion 1985] for one approach to this transformation) defining the elemental detail of the model. Figure 3-5 shows the elemental detail relations for the transportation model.

PLANT(plant_id, supply, test:sup, interpretation)

CUST(cust_id, demand, test:dem, interpretation)

LINK(plant_id, cust_id, cost, flow, interpretation)

Figure 3-5: Relational Form of Elemental Detail for Transportation Model

Perhaps the most appealing aspect of structured modeling is this coordination of models and data. Defining the model automatically defines the data requirements as a by-product. The data can then be queried in conjunction with the model schema and vice versa. This provides a high degree of flexibility not found in most modeling systems.

3.3 ORACLE Implementation

A preliminary version of an RIRDS which supports SM is under development as part of this project. The current version operates in passive mode but will eventually use the IRDS in active mode as it evolves into a more fully functional model management system. The system is implemented on a Vax 780 under the VMS

operating system using the ORACLE RDBMS. The ORACLE tables for the RIRDS are presented in Appendix D and sample Fortran code for the MMS is displayed in Appendix E. Efforts are currently underway to migrate to a PC environment using PC-ORACLE with a C language interface.

4. JACKSON SYSTEM METHODOLOGIES

There are two aspects to the Jackson approach to information system development: Jackson System Development (JSD) and Jackson Structured Programming (JSP). JSD is essentially a discrete event simulation approach to building information systems whereas JSP is a programming convention which facilitates the detailed definition and structure of the processes identified by JSD. Although we have artificially separated the two in order to evaluate their compatibility with structured modeling, in practice, building an information system requires the coordinated use of both JSD and JSP.

4.1 Jackson System Development and Structured Modeling

The investigation of the relationship between JSD and SM was subcontracted to, and carried out by, Professor Jeffrey Kottemann from the University of Hawaii at Manoa. Most of the material contained in this section (and related subsections) is a distillation of his report [Kottemann 1986].

4.1.1 A brief survey of JSD

Because the Jackson methodology has been developed and used primarily in the United Kingdom and Europe, there is relatively little documentation available in this country. Cameron's overview article on JSD [Cameron 1986] is used as the source for the following discussion.

In the JSD world, building an information system is roughly equivalent to building a discrete event simulation model based on the notion of communicating sequential processes (CSP) [Hoare 1978]. JSD is divided into three phases: the Model phase, the Network phase, and the Implementation phase.

In the Model phase, JSD focuses on **events** performed by, or upon, **entities**. A **process** pertains to actions of a specific entity assuming a particular **role**. A **process model** depicts the time-ordered actions of a **single occurrence** of an entity and is represented as a Jackson Structured Program using the regular expressions of structured programming. The nested ordering of events in the JSP implicitly defines the allowable series of events that may occur to the entity associated with this process (Figure 4-1).

In the Network phase, processes are linked using **data stream** and **state vector** connections as in Hoare's CSP. With a data stream, communication is initiated by the source process which supplies the data. With a state vector, the recipient process calls the source process to access its (the source's) data. Each state vector contains a variable indicating where in the process structure the process was last active.

```
LOANPART itr while (inputaction = LEND)
  LOAN seq
    LEND seq
      read next inputaction;
    LEND end
    OUT-ON-LOAN itr while (inputaction = RENEW)
      RENEW seq
        read next inputaction;
      RENEW end
    OUT-ON-LOAN end
    RETURN seq
      read next inputaction;
    RETURN end
  LOAN end
LOANPART end
ENDPART sel (inputaction = SELL)
  SELL seq
  SELL end
ENDPART alt (inputaction = OUTCIRC)
  OUTCIRC seq
  OUTCIRC end
ENDPART end
```

Figure 4-1: Process Description for Cameron's Book Example

Processes and embedded functions related to input/output activities and datastream merging are also added to the system in this phase. JSD logically separates the model processes from the I/O functions which is common practice in discrete event simulation (e.g.: [Zeigler 1984]).

In the Implementation phase, the process network is organized into a run unit via the **inversion** mechanism. Inversion is probably the most counterintuitive aspect of JSD since it relies on the notion of coroutines and, in some cases, violates conventional structured programming practices. The main purpose of inversion is to convert processes into subroutines which are then called by a master scheduler (roughly equivalent to an event chain in other discrete event environments). Inversion also implements data streams (i.e., message passing) using traditional call-and-return mechanisms and resolves **clashes** which result from mismatches between physical input and output structures. (See [Kottemann 1986] for more details on inversion.)

4.1.2 Representing JSD in structured modeling

The relationship between JSD and SM can be viewed in two ways: with SM acting either in a passive or active mode. In the

passive mode, SM serves primarily as a representation medium to capture JSD models. In an active mode, SM would somehow define its own discrete event environment which JSD would then have to conform to. The difference between these approaches is subtle and can be confusing at times. We will discuss the passive mode in this section and the active mode in the next section.

In the passive mode, JSD would be done exactly as its proponents describe, only the resultant JSD diagrams and JSP processes would then be transformed into SM notation and subsequently included in the IRDS as described in Section 3. Thus the modeler would specify a JSD diagram such as that in Figure 4-2 for the situation "when a book is returned check if there are any reservations outstanding for that title and output the name and address of the member who has been waiting the longest." This would then be transformed into an SM schema. Figure 4-3 shows one possible schema which could be used to represent all JSD diagrams. Elemental detail is shown for the BOOK example which Cameron uses in his article [Cameron 1986]. The SM generic structure for this schema is shown in Figure 4-4.

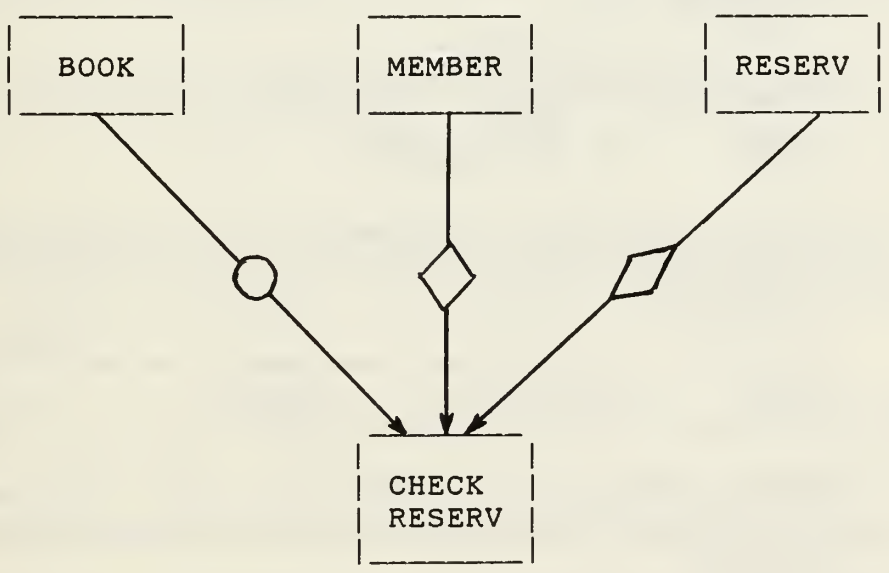


Figure 4-2: Partial JSD Diagram for Cameron's Book Example

The reason we characterize SM as passive in this scenario is that SM really is contributing nothing conceptually to the JSD modeling process. It is acting only as a representation medium which, in fact, could be made totally transparent to the modeler. The modeler could do JSD without knowing anything about SM. Although this indicates that JSD can be done within our SM/IRDS

&JSD There is a JSD simulation model.

EVENTe /pe/ A list of EVENTS in the simulation.

EVENT Ident.	Interp.
LEND	Lend a book
RENEW	Renew a book
RETURN	Return a book
SELL	Sell a book
OUTCIRC	Remove book from circulation

ENTITYi /pe/ A list of ENTITIES in the simulation.

ENTITY Ident.	Interp.
BOOK	Library book
MEMBER	Library member

ATTRIBUTEa /pe/ A list of ATTRIBUTES in the simulation.

ATTRIBUTE Ident.	Interp.
MEMBER_ID	Member identification no.
NAME	Member name.
ISBN	Book ISBN.

POSSESSES(ENTITYi,ATTRIBUTEa*(i)) /ce/ Each ENTITY POSSESSES a list of associated ATTRIBUTES.

POSSESSES Entity	Attr1	Attr2	Attrn
BOOK	ISBN	TITLE	
MEMBER	MEMBER_ID	NAME	ADDRESS

PROCESSp(ENTITYi1(p),EVENTe*(p)) /ce/ A process depicts the time-ordered EVENTS of a single occurrence of an entity.

PROCESS Ident.	Entity	Event1	Interp.
LOANPART	BOOK	LEND RENEW RETURN	Loan a book
ENDPART	BOOK	SELL OUTCIRC	Dispose of bk

Figure 4-3: SM Schema for General JSD Model (Cont'd)

DATASTREAMd(PROCESSp1(d),PROCESSp2(d)) /ce/ Source process (p1) initiates communication with recipient process (p2) by supplying data.

DATASTREAM Ident.	Source_Proc	Recip_Proc	Interp
DS01	BOOK	CHECK RESERVE	Check for waiting list for returned book

STATEVECTORs(PROCESSp3(s),PROCESSp4(s)) /ce/ Recipient process (p4) calls source process (p3) to access p3's data.

TEXTPOINTER(STATEVECTOR(s)) /a/ :I+ Program counter.

STATEVECTOR Ident.	Source_Proc	Recip_Proc	Text_Ptr	Interp
SV01	RESERVATION	CHECK RESERVE	1	Check reservation
SV02	MEMBER	CHECK RESERVE	1	Longest waiting member

Figure 4-3: SM Schema for General JSD Model

environment, it begs the larger question of the coexistence of two different modeling idioms which the modeler must contend with. This is likely to engender confusion rather than enlightenment.

What needs to be considered then is whether there is some way to represent the JSD diagrams such as Figure 4-2 directly as SM generic structures.

4.1.3 Structured modeling and discrete event simulation

The major problem in deciding whether SM is applicable to discrete event models is the lack of a notion of what a solver entails in this environment. For examples like the transportation model, solvers exist in the form of simplex algorithms. Discrete event equivalents, however, really don't exist except in the form of programs (e.g.: SIMSCRIPT) which execute particular simulations. Thus, we cannot assume that system dynamics can be handled by a solver; instead, the model definition must include the dynamics.

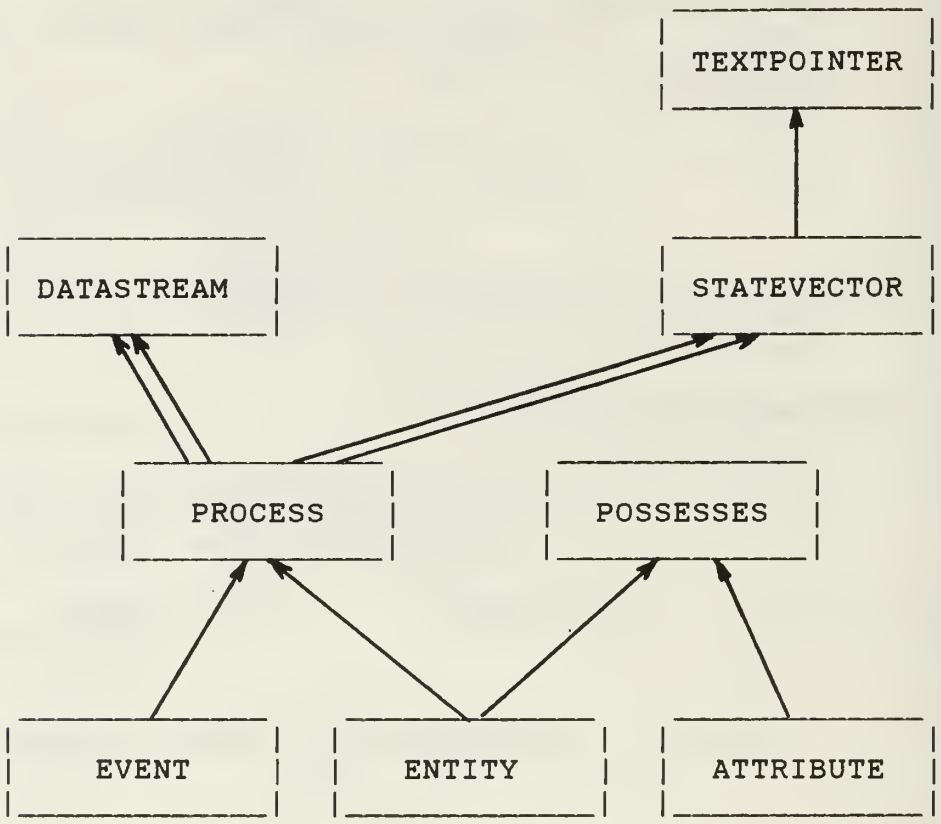


Figure 4-4: SM Generic Structure of JSD Model

One of the major problems then in applying SM to discrete event simulation is the representation of states and the conditions under which state transition takes place. Kottemann indicates how SM might use test elements to define legal states into which the model may move but the time element and state transition functions are not addressed.

The representation of time in SM is an undecided issue. Although Geoffrion presents a control-in-continuous-time problem in his monograph [Geoffrion 1985, pp. 2-91,92] which explicitly represents time, he admits that he violates one of the assumptions of SM by introducing a genus with an infinite number of elements. This can be circumvented in the discrete event case since we can assume a finite number of time instants of interest but the problem of how one represents the equivalent of an events chain still looms.

Geoffrion presents an intriguing scenario regarding SM in support of discrete event simulation. He suggests that a static

structured model be prepared of the system to be simulated and then a control program composed which edits the elemental detail tables according to the rules governing the system's dynamic behavior. The control program would use the dynamics described in a **dynamic** part of the schema which complements the static part. The control program would also accept directions from the users in a nonprocedural language defining the nature of the simulation experiment to be performed. These directions, or commands, can be thought of as specifying a task to a solver in connection with a specific model.

Geoffrion's idea sounds similar in many respects to SIMSCRIPT as he admits:

"Thus, the control program would in effect become a kind of all-purpose solver for discrete event simulation in the context of structured modeling. It would not need to be customized for each application. **No such solver has yet been built, and it is not obvious whether the idea is practical.** It is encouraging to observe that SIMSCRIPT can be viewed as working according to a similar plan..." [Geoffrion 1986b, p. 17]

4.1.4 Summary of interplay between JSD and structured modeling

JSD captures the dynamic aspects of a simulation model whereas SM captures the static aspects. JSD deals with logical relationships by applying inversion whereas SM has no known mechanisms at present for representing dynamic properties. JSD can be implemented on top of an SM environment using SM as a representation medium but this is not really a "marriage" of the two approaches. JSD and SM can be "married" (at least conceptually) as shown by Kottemann with a result which is similar to the environment of Simscript II.5. Nevertheless, the alien cultures of JSD and SM would require an immense learning effort to use both in conjunction.

4.2 Jackson Structured Programming and Structured Modeling

The investigation of the relationship between JSD and SM was subcontracted to, and carried out by, Professor Jack Stott from the University of Hawaii at Manoa. Most of the material contained in this section (and related subsections) is a distillation of his report [Stott 1986].

4.2.1 A brief survey of JSP

JSP is used to represent JSD process models via the regular expressions of structured programming: sequence, selection, and iteration. **Sequence** defines the time-ordered occurrence of actions, thus if actions A and B follow in a sequence, action B must **always** follow action B. **Selection** defines **either** action A or action B occurring. **Iteration** defines one or more actions occurring repeatedly in the order defined by the sequence. A

process description may be comprised of nested levels of blocks containing any of the three types of expressions (see Figure 4-1). These descriptions can be represented as trees where sequence goes from left to right, selection is denoted by an "o" in a box, and iteration is denoted by a "*" in a box (Figure 4-5).

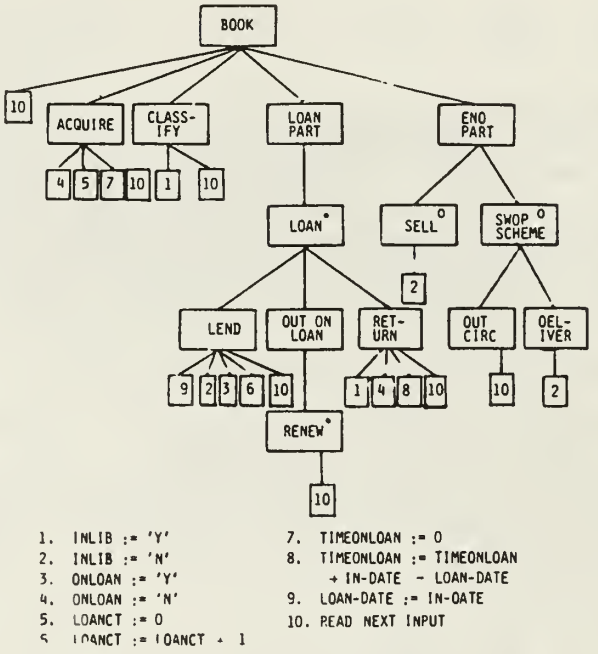


Figure 4-5: Process Tree for Cameron's Book Example

4.2.2 Representing JSP in structured modeling

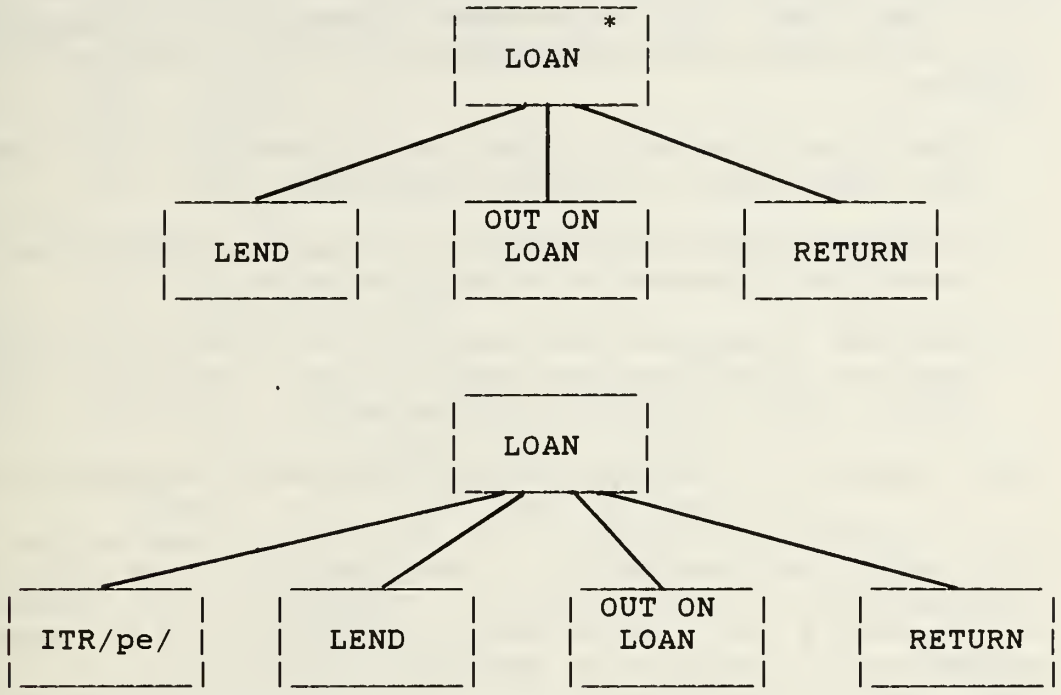
A JSP structure like Figure 4-5 can be cast as a "quasi"-structured model in the following way:

1. Decompose selection and iteration boxes by forming a sequence which starts with a separate selection or iteration box, respectively (Figure 4-6).

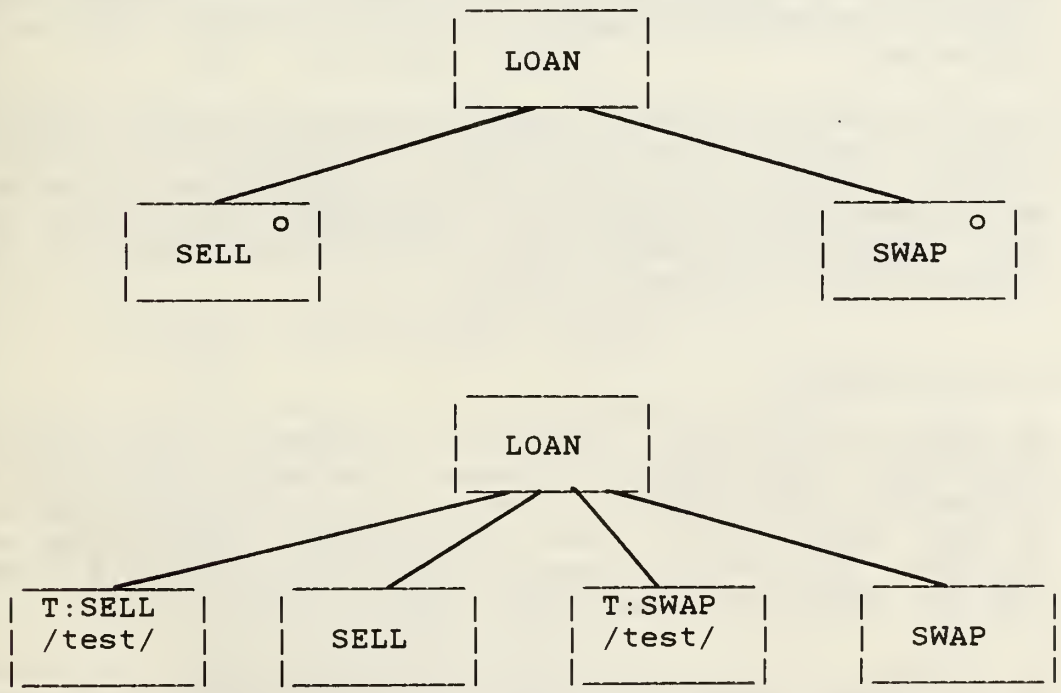
2. Consider all boxes with offspring as modules.

3. Consider all leaf boxes as genera.

4. Elemental detail consists of all statements (the numbered boxes in Figure 4-5) associated with a particular genus.



(a) SM representation of JSP iteration.



(b) SM representation of JSP selection.

Figure 4-6: Decomposing Iteration and Selection in SM

Applying this to Figure 4-5 yields the model schema in Figure 4-7. Note that this corresponds almost directly to Figure 4-1. Unfortunately, this falls short of being a coherent SM for two reasons:

1. There is no generic structure to speak of since the genera by and large have no calling sequences.

2. Elemental detail is trivial in that each genus has only a few executable statements associated with it.

3. Modular structure and elemental structure must be ordered to capture the time-dependent nature of events in the Jackson world. This is not a feature of SM.

4.2.3 Summary of interplay between JSP and structured modeling

SM is primarily a logical, and therefore nonprocedural, framework for model representation. Programming in JSP, and any other language, is, on the contrary, a very procedural undertaking. It is therefore not surprising that there does not seem to be a convenient fit between SM and JSP. Although, at first blush, there seem to be constructs in SM which correspond to the basic structured programming expressions (e.g.: calling sequence equivalent to sequence, primitive entity equivalent to iteration, and test entity equivalent to selection), a program does not seem to have sufficient generic structure to form a robust SM equivalent.

One alternative approach might to represent everything in Figure 4-5 as generic structure but this results in a model with no modular structure and therefore no indentation which is so vital to block-oriented languages. Further, this gives rise to some very suspicious genera for which it is difficult to imagine the corresponding elemental detail.

4.3 Conclusions

SM does not seem well adapted to representing process-oriented or procedural phenomena. Although SM can be used as a passive medium for both JSD/JSP, there is no real advantage to doing so. On the other hand, given the peculiar conventions adopted by JSD/JSP, there does not seem to be a convenient fit where SM and JSD/JSP can coexist usefully.

It seems unlikely that one could be a productive modeler if he had to learn the terminology and conventions of both disciplines. In a discrete event environment, it would seem that JSD/JSP and SM present an either/or situation. Either adopt JSD/JSP as the accepted methodology or search for ways to strenghten SM as a discrete event tool. Recommendations for resolving this dilemma are presented in the Conclusions (Section 6).

```
&BOOK  Book module.
  ACQUIREa /pe/ Book acquisition statements.
   s4    ONLOAN := 'N'
   s5    LOANCT := 0
   s7    TIMEONLOAN := 0
   s10   READ-NEXT-INPUT

  CLASSIFYc /pe/ Book classification statements.
   s1    INLIB := 'Y'
   s10   READ-NEXT-INPUT

  &LOANPART  Book loaning module.
    ITR /pe/ Iteration.
          WHILE INPUTACTION = 'LEND'
    &LOAN  There be many book loans.
      LEND1 /pe/ A book must be lent 1st.
       s9    LOAN-DATE := IN-DATE
       s2    INLIB := 'N'
       s3    ONLOAN := 'Y'
       s6    LOANCT := LOANCT + 1
       s10   READ-NEXT-INPUT

      &OUTONLOAN  Books may be out on loan.
        ITR /pe/ Iteration.
              WHILE INPUTACTION = 'RENEW'
        RENEWr /pe/ A book may be renewed many times.
         s10   READ-NEXT-INPUT

      RETURNq /pe/ A book may be returned after it's lent.
       s1    INLIB := 'Y'
       s4    ONLOAN := 'N'
       s8    TIMEONLOAN := TIMEONLOAN + IN-DATE - LOAN-DATE
       s10   READ-NEXT-INPUT

    &ENDPART  A book may be disposed of.
      T:SELL /test/ input = SELL
       true
      SELL(T:SELL) /ce/ A book may be sold.
       s2    INLIB := 'N'

      T:SWAP /test/ input = SWAP
       true
      &SWAP  A book may be swapped.
        OUTCIRC(T:SWAP) /ce/ Throw book away.
         s10   READ-NEXT-INPUT

        DELIVER(T:SWAP) /ce/ Swap book.
         s2    INLIB := 'N'
```

Figure 4-7: SM Representation of JSP Book Process

5. ONEC MODEL AS A STRUCTURED MODEL

One of the major objectives of the first phase of this project is to try to represent an existing combat simulation model as a structured model. The model chosen was the ONEC "Fight the Battle" which is a segment of the larger FOURCE simulation. ONEC was selected because it's one of the simpler models in TRASANA's inventory and has already been implemented as a Fortran program.

The attempt to cast ONEC as a structured model has been done in conjunction with Captain David Patrick, USAF, as part of his Master's thesis at Naval Postgraduate School. The ONEC source document we've used is [TRASANA 1978]. Our progress in this aspect of the project has been unexpectedly slow for two reasons. First, neither of us has had any appreciable exposure to combat or simulation models, so much of the terminology and many of the concepts are unfamiliar to us. Second, this is our first attempt to apply structured modeling to a complex problem and we've encountered many situations where we've had to search painstakingly for the appropriate SM constructs to model these situations. Whether this is a problem inherent with SM or just another manifestation of our modeling naivete has yet to be determined. Nevertheless, the confluence of two steep learning curves has resulted in only a partial model rather than the full SM we had anticipated.

5.1 Generic Structure

We have concentrated primarily on trying to develop a complete generic structure for ONEC. We've arbitrarily divided the generic structure into three segments corresponding to movement (position and speed), combat support, and direct-fire engagements. Our initial focus has been on the movement segment, the generic structure for which is shown in Figure 5-1. Areas where confusion still exists about SM constructs are denoted by question marks in the model schema below.

5.2 Modular Structure

The modular structure is particularly vital for a model this complex since the generic structure otherwise becomes too cluttered to understand. Figure 5-2 shows a modular structure for ONEC as we interpret it. We have divided the model into four major modules representing the battlefield layout, troop movement, combat support, and direct fire engagement respectively. At this stage we have concentrated our efforts on the battlefield and movement so only those two modules are developed further. Note that when we begin to develop the combat support and direct fire engagement portions of the model more fully, these branches of the modular tree will be expanded accordingly. This top-down approach to model building is one of the attractive features of SM.

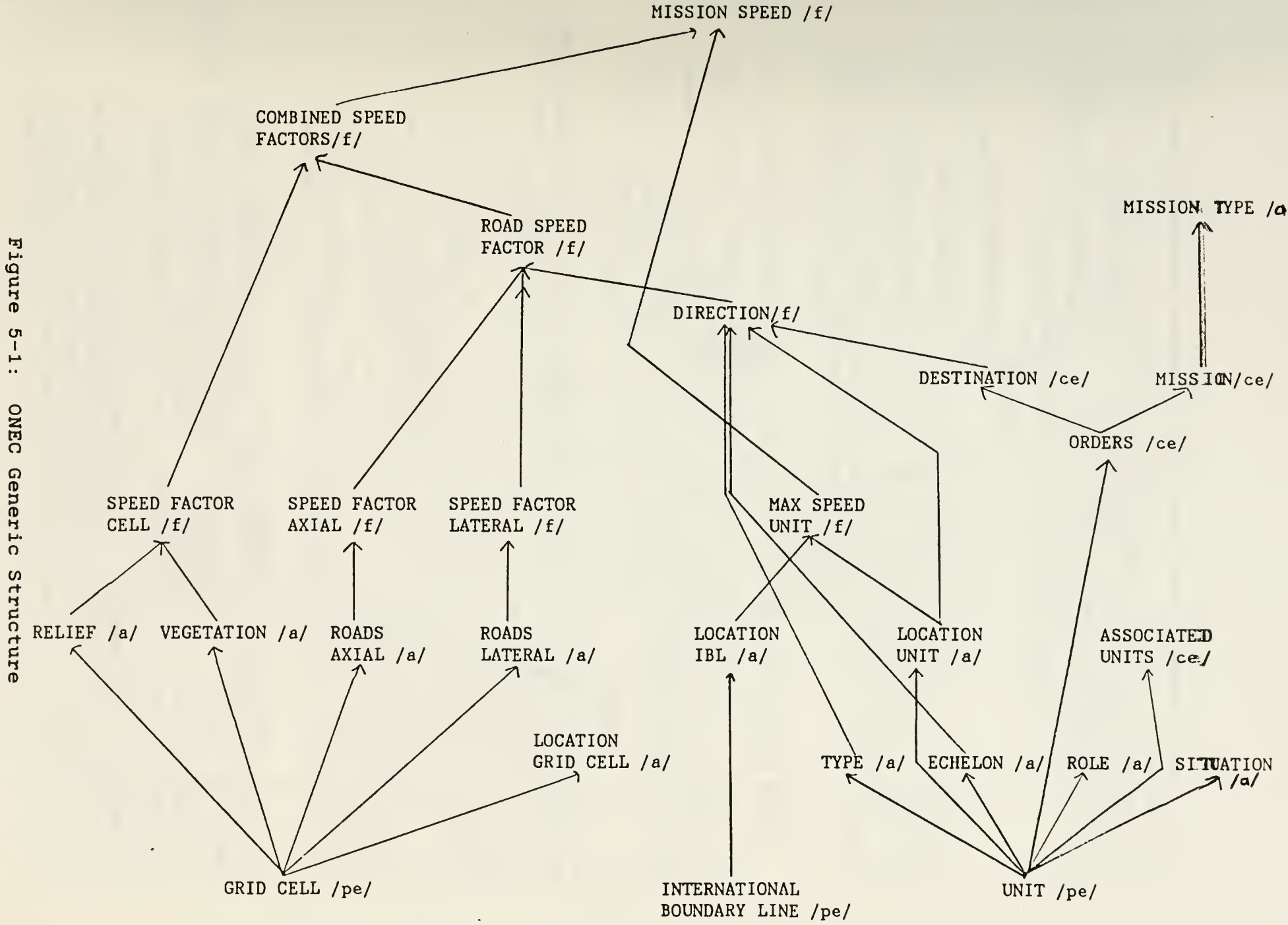


Figure 5-1: ONEC Generic Structure

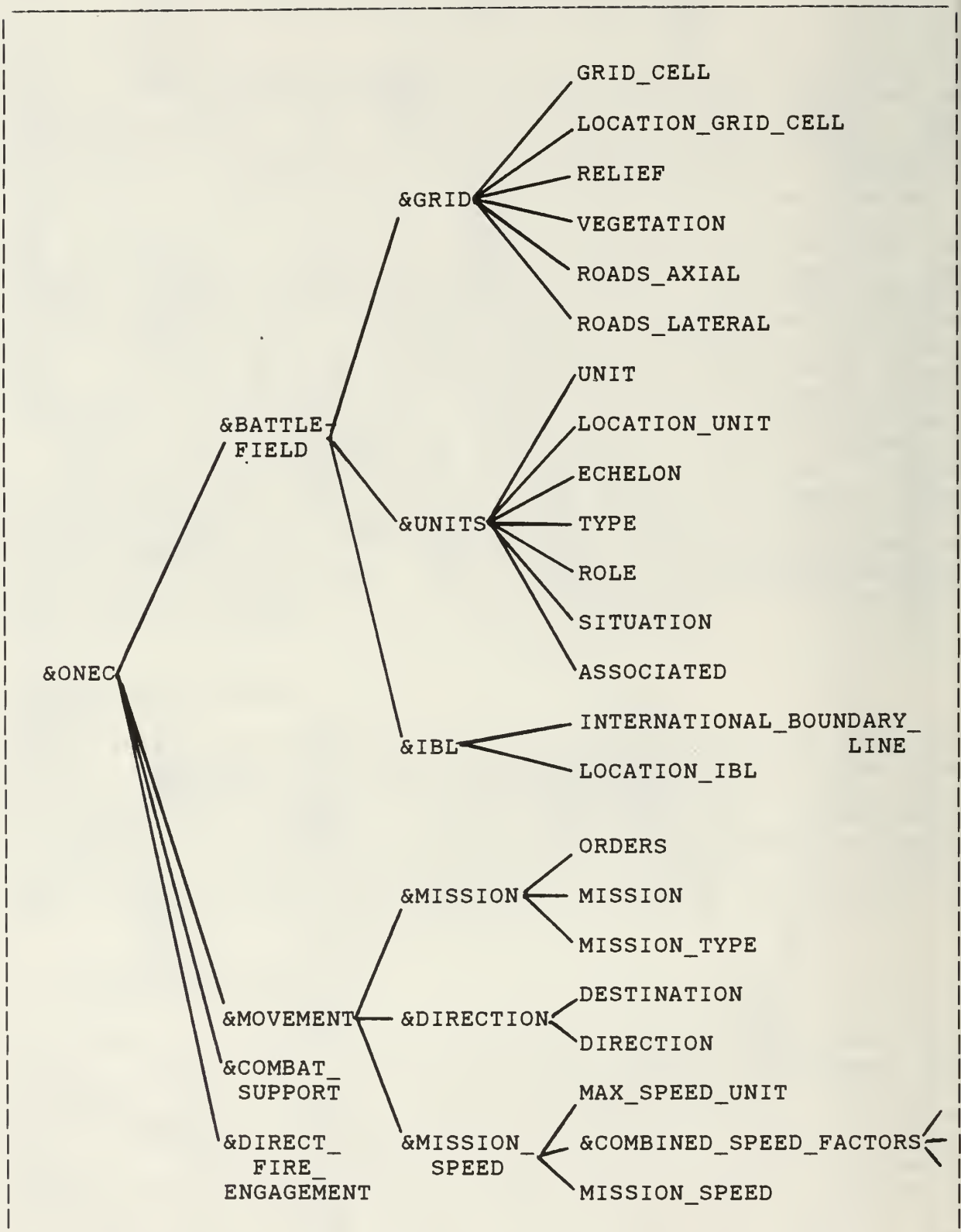


Figure 5-2: Modular Structure for ONEC Model

The model schema corresponding to the generic and modular structures above is shown in Figure 5-3. Again, we have left combat support and direct fire engagement for later expansion. By defining the modules in this manner, we can now display genus graphs of the model at various levels of abstraction as shown in Figure 5-4. Of course, each module can be exploded to any desirable level of existing detail in these graphs.

5.3 Elemental Structure

Since the elemental structure (elemental detail) is directly derived as a set of relational tables from the generic structure, it too depends on completion of the generic structure. Once the full generic structure is developed, it will be necessary to generate appropriate test data with which to fill these tables. Data generation is TRASANA's responsibility.

5.4 Problems Encountered in Building the ONEC Structured Model

Geoffrion's publications on SM present a multitude of relatively simple models cast in structured modeling form. This is somewhat deceptive because it does not address the process of casting these models into this logical representation. Upon trying to model a more complex environment, we found that SM offers substantial complexities of its own:

1. The genus graphs hide much of the nitty-gritty semantics necessary to fully understand a model. The calling sequences and index-set statements comprise the real semantics and it is often very difficult to determine how to represent them. In particular, we found that modelers must become very adept at manipulating indexes in order to represent logical relationships. A rigorous syntax based on relational algebra or first order predicate calculus, for example) must be devised to represent these statements. Geoffrion is currently developing such a syntax. The mathematical nature of SM becomes very evident as one proceeds in building large models. It is clear from our experience that, without software support, **users of SM must be very sophisticated modelers.**

2. Basic logical relationships such as hierarchies (one-to-many), generalizations, inheritance, and recursivity are not always straightforward to represent in SM. There are many different strategies for representing certain logical relationships and it is not clear which one is preferable in any particular situation. Although this is characteristic of any modeling approach, **a set of heuristics should be derived to aid the structured modeler in representing frequently encountered logical relationships.** Keep in mind that SM is very new and has very few practitioners at this point. It is reasonable to suppose that heuristics will evolve as more users adopt SM.

&ONEC The ONEC component of the FOURCE combat simulation.

&BATTLEFIELD The BATTLEFIELD setup.

&GRID The BATTLEFIELD is represented as a GRID.

GRID_CELLg /pe/ The GRID consists of 1610 1km x 3km GRID_CELLs placed on a 35km x 138km BATTLEFIELD with their long sides parallel to the long side of the BATTLEFIELD.

LOCATION_GRID_CELL(GRID_CELLg) /a/ {GRID CELL} : (x,y) coordinate pair. Every GRID_CELL has a LOCATION expressed as the (x,y) coordinate of the southwest corner of the GRID_CELL.

RELIEF(GRID_CELLg) /a/ {GRID CELL} : (5Dd,5Dc,5Ec,5Fc) Each GRID_CELL has relief as indicated by the four possible configurations on the Natick Landform Classfication Code.

VEGETATION(GRID_CELLg) /a/ {GRID CELL} : (0..10) Each GRID_CELL has a value associated with it that tells the fraction of the cell covered by vegetation.

ROADS_AXIAL(GRID_CELLg) /a/ {GRID CELL} : (none, primary only, secondary only, primary and secondary) Each GRID_CELL has a value for ROADS in the AXIAL direction.

ROADS_LATERAL(GRID_CELLg) /a/ {GRID CELL} : (none, primary only, secondary only, primary and secondary) Each GRID_CELL has a value for ROADSin the LATERAL direction.

&UNITS UNITS are situated on the BATTLEFIELD.

UNITu /pe/ There are many types of UNITS participating in this simulation.

LOCATION_UNIT(UNITu) /a/ {UNIT} : ((x1,y1), (x2,y2)) Every UNIT has a location expressed as two pairs of coordinates corresponding to its southwestern point (x1,y1) and its northeastern point (x2,y2).

ECHELON(UNITu) /a/ {UNIT} : (First, Second, Reserve); Every UNIT has an ECHELON.

Figure 5-3: ONEC Model Schema
(Cont'd)

TYPE(UNITu) /a/ {UNIT} : (Division,Regiment,Battalion, Battery) ; Each UNIT has a TYPE. There is a hierarchy of units. Divisions are composed of Regiments which are composed of Battalions which are composed of Batteries.

ROLE(UNITu) /a/ {UNIT} : (????) It is not clear what is meant by ROLES. Documentation states that ROLES are required to find direction.

SITUATION(UNITu) /a/ {UNIT} : (????) This might need to be further refined. Somewhere we must account for combat situation.

ASSOCIATED(UNITu1,UNITu2) /ce/ Some UNITS move directly with other UNITS [TRASANA 1978, p.5-16].

&IBL There is an INTERNATIONAL BOUNDARY LINE.

INTERNATIONAL_BOUNDARY_LINE /pe/ There is a line called the INTERNATIONAL_BOUNDARY_LINE. It separates the FRIENDLY side from the ENEMY side and is used to determine the maximum speed of a UNIT.

LOCATION_IBL(INTERNATIONAL_BOUNDARY_LINE) /a/ {INTERNATIONAL_BOUNDARY_LINE}: (y coordinate); There is an IBL on the map. It is described as a straight line. The exact location is the the Y coordinate of the line.

&MOVEMENT UNITS have MOVEMENT on the BATTLEFIELD.

&MISSION UNITS are assigned MISSIONS.

ORDERS(UNITu) /ce/ ORDERS are used to give UNITS their MISSION and DESTINATIONS. Each UNIT must have a set of ORDERS.

MISSION(ORDERSu) /ce/ Each set of ORDERS includes a MISSION.

MISSION_TYPE(MISSIONu) /a/ {MISSION} : (RED attack, RED holding attack, RED be prepared to attack, BLUE defend, BLUE delay, BLUE withdraw, BLUE reserve, BLUE move to reinforce) ; There are specific TYPES of MISSIONS.

Figure 5-3: ONEC Model Schema
(Cont'd)

```
&DIRECTION  Moving UNITS have specified DIRECTION.

  DESTINATION(ORDERSu) /ce/ Each set of ORDERS includes a
   DESTINATION.

  DIRECTION(LOCATION_UNITu,DESTINATIONu,TYPEu,ECHELONu)
   /f/ DIRECTION = SQRT((DESTINATION - LOCATION_UNIT)**2)
   Each moving UNIT must have a DESTINATION and a subse-
   quent DIRECTION.

&MISSION_SPEED  Each UNIT in MOVEMENT has a MISSION_SPEED

  MAX_SPEED_UNIT(LOCATION_UNITu,INTERNATIONAL_BOUNDARY_
   LINE) /f/ If  LOCATION_UNITu = FRIENDLY SIDE of
   INTERNATIONAL_BOUNDARY_LINE then MAX SPEED UNITu =
   25Km/Hr else MAX SPEED UNITu = 15Km/Hr.

  &COMBINED_SPEED_FACTORS  There are many factors which
   can reduce the MAX_SPEED_UNIT.

    SPEED_FACTOR_CELL(RELIEFg,VEGETATIONg) /f/
     {GRID_CELL}      : 1 ;  Each GRID_CELL has a
     maximum speed factor based on the RELIEF and
     VEGETATION.  This is found by a table look-up
     [TRASANA 1978, Pg. 5-9, Table 5-2].

    SPEED_FACTOR_AXIAL(ROADS_AXIALg) /f/ {GRID_CELL} :
     ??Index_Set_Statement?? ; Each GRID_CELL has a
     maximum speed factor in the axial direction based
     on  the  types  of ROADS present.  This is a table
     look-up and generates a fraction of speed allowed
     factor [TRASANA 1978, Pg. 5-9, Table 5-3].

    SPEED_FACTOR_LATERAL(ROADS_LATERALg) /f/ {GRID_CELL}
     ??Index_Set_Statement?? ; Each GRID_CELL has a
     maximum speed factor in the lateral direction  based
     on the types of roads present.  This is a table
     look-up and generates a fraction of speed allowed
     factor [TRASANA 1978, Pg. 5-9, Table 5-3].

    ROAD_SPEED_FACTOR(SPEED_FACTOR_AXIALg1(u),SPEED_
      FACTOR_LATERALg1(u),DIRECTIONu) /f/ {?????} :
      ??Index_Set_statement?? ; ROAD SPEED FACTOR =
      [TRASANA 1978, Equation 5-7 on Pg. 5-10].
      This combines the speed factors in the axial  and
      lateral directions for a single speed factor which
      takes into account the direction of travel and the
      roads.
```

Figure 5-3: ONEC Model Schema (Cont'd)

```
COMBINED_SPEED_FACTORS(SPEED_FACTOR_CELLg,ROAD_SPEED_
 FACTORg) /f/ {GRID_CELL} : ??Index Set Statement?? ;
 COMBINED_SPEED_FACTOR = SPEED_FACTOR_CELLg *
  ROAD_SPEED_FACTOR  This combines all of the speed
 factors involved in a GRID_CELL and a UNIT in that
 GRID_CELL into a single COMBINED_SPEED_FACTOR.

MISSION_SPEED(COMBINED_SPEED_FACTORSg,MAX_SPEED_UNITu)
   /f/ MISSION_SPEED = MAX_UNIT_SPEED  *
         COMBINED_SPEED_FACTORS
   Overall MISSION_SPEED for a UNIT in MOVEMENT.

&COMBAT_SUPPORT  UNITS have COMBAT_SUPPORT on the BATTLE-
  FIELD.

&DIRECT_FIRE_ENGAGEMENT  UNITS have DIRECT_FIRE_ENGAGEMENT
  with enemy UNITS.
```

Figure 5-3: ONEC Model Schema

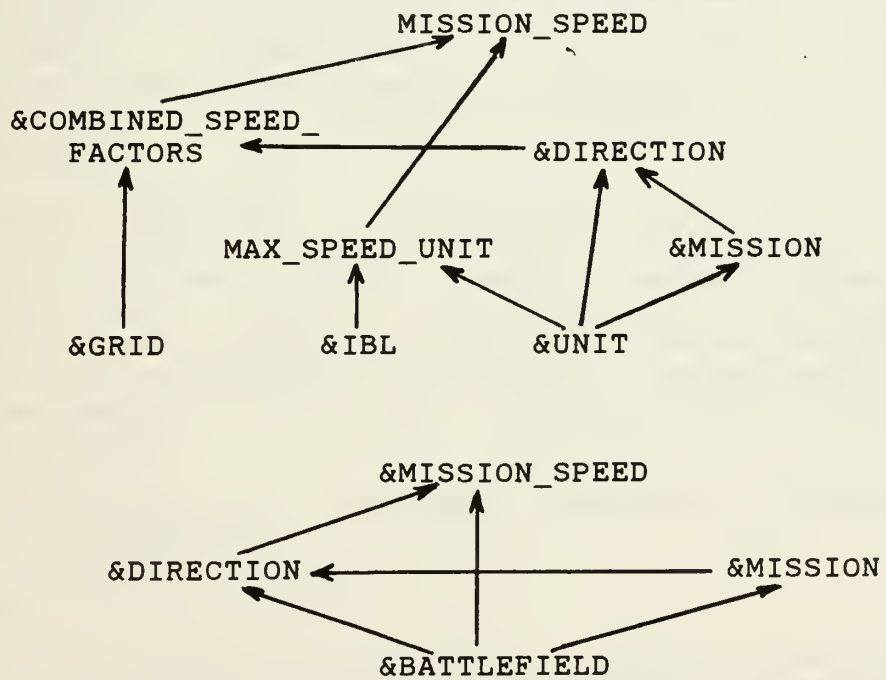


Figure 5-4: Alternative ONEC Genus Graphs

5.5 Summary

Our experience as summarized from the section above suggests that there is a **significant learning curve for SM** initially. This should not be underestimated since it may translate into **substantial training** costs for any organization wishing to adopt SM. Note however that the same situation is true for equivalent methodologies such as JSD. One advantage of SM in this regard is that there is likely to be an expanding body of accessible expertise as the discipline grows.

We approached the SM representation of ONEC from the bottom up in the sense that we tried to derive the genus graphs first. This proved to be very difficult and confusing. Only when we began to impose modular structure did things begin to fall in place. This points out the need for deriving good modeling practices and methodologies associated with SM and for providing software to support these techniques as well. It should be mentioned that we did not use any software tool in our model development. It is reasonable to expect that an implemented SM system such as described in Geoffrion [Geoffrion 1985] may have facilitated our efforts considerably.

6. CONCLUSIONS

The following things have been accomplished in this phase of the project:

1. The design and initial prototype implementation of a FIPS-compatible information resource dictionary system on the ORACLE DBMS;

2. The extension of the IRDS to accommodate structured modeling representations;

3. A comparison of structured modeling and JSD/JSP with respect to their suitability for combat simulation modeling.

Previous sections have described the operation of the dictionary system. We conclude by reviewing and comparing the two disciplines and making recommendations for the second phase of this project.

6.1 Jackson System Methodology and Structured Modeling

JSD/JSP is a discrete event simulation approach to information systems development which is based on the notion of communicating sequential processes [Hoare 1978]. It is process-oriented in that it focuses on the description of time-ordered events which are performed upon single entities. The need for these processes to communicate with one another determines the nature of data requirements. Implementation of a JSD/JSP system requires extensive use of the inversion technique for facilitating this communication. Thus, JSD/JSP is a dynamic, rather than a static, methodology.

Structured modeling, on the other hand, is a more general approach to modeling "in the large". It provides a framework for representing the static, logical relationships between the components of a model. Its three basic structures (generic, modular, and elemental) facilitate cross-referencing, model views, and model-data relationships respectively. Structured modeling also promotes model integration and top-down or bottom-up development.

The following points illuminate the primary differences between the two disciplines:

1. Underlying formalism

   JSD/JSP is more of a methodology than a formalism although it is based on the notion of communicating sequential processes. SM is based on a complete, mathematical formalism. Like the relational theory for databases, this should promote continued research and development of SM.

2. Support of discrete event simulation

   JSD/JSP is dynamic and process-oriented and therefore well-suited to discrete event simulation. SM is static and therefore needs additional features to support discrete event modeling.

3. Programming environment

   JSD/JSP has a built-in programming environment in the form of JSP. Programming relies heavily on the counterintuitive notion of inversion and is done largely bottom-up. SM currently has no associated programming environment with which to build a program which can run a simulation described in SM. This must be developed.

4. Documentation and communication

   JSD/JSP provides diagrams and graphs as documentation aids. JSD/JSP does not support any form of abstraction, however. SM provides graphs, schemas, and a wide range of views. SM is a better tool for model documentation and communication.

5. Management control

   JSD/JSP has a limited dictionary capability which restricts resource sharing and management control over the modelbuilding process. SM has a full FIPS-compatible dictionary developed in conjunction with this project.

6.2 Structured Modeling or JSD/JSP?

SM and JSD/JSP are in one sense complementary in that SM is good for representing the static aspects of a model and JSD/JSP is good for representing the dynamic aspects. The question arises then, why not use both in developing a modeling environment?

There are two basic responses which argue against this approach. The first is that the two disciplines are not really complementary. JSD/JSP is predicated upon inversion as a means of representing logical relationships between entities and cannot really accommodate an SM approach. Second, and more important, each discipline has its own unique vocabulary and concepts, little of which overlaps. Further, each approach is complex in its own right. The learning curve for each is steep and the learning curve for the two in conjunction is essentially additive and therefore even steeper. Both approaches require sophisticated modelers; attempting to meld the two environments into one would result in prohibitive complexity which would undermine the eventual utility of the model management system.

A decision must be made to adapt one of these disciplines as the baseline environment and build the model management system on that foundation. This will reduce the complexity to a manageable

level and provide a unified approach to modeling.

JSD/JSP is a known quantity, having been around for 10 years. If JSD/JSP is selected, this is tantamount to proclaiming that TRASANA will become a JSD/JSP shop. Possible disadvantages to this choice include the counterintuitive programming principles which JSD/JSP espouses plus the relative inaccessibility of expertise in this country. Although the dictionary system developed herein can be adapted for use in a JSD/JSP environment, there is no need to continue this research project since training and implementation of the methodology can be done externally (using Jackson personnel, for example).

As an alternative to adopting JSD/JSP, further research can be done to investigate strengthening the links between SM and discrete event simulation. This offers the dual advantages of providing a potentially powerful alternative to JSD/JSP tailored to TRASANA's needs, and an already existing dictionary system for performing model documentation.

6.3 Recommendations

We recommend the following approach:

1. Phase 2 of the project should be restated to emphasize research in the area of strengthening the link between structured modeling and discrete event simulation. In particular, a promising approach would be to consider incorporating aspects of Zeigler's ideas about discrete event simulation into SM [Zeigler 1984, Kottemann 1986].

2. Phase 2 of the project should still include refinements to the prototype model management system, especially graphics, but this should now assume a secondary role.

3. A decision should be made at the end of Phase 2 concerning the feasibility of SM as a tool for discrete event simulation in general, and combat modeling in particular. If the decision is negative, then the project should be terminated at the end of Phase 2.

4. If Phase 2 is successful, Phase 3 of the project should be directed towards developing programming environments which use structured modeling as their basis. This would allow modelers to build programs to run simulations which are represented as structured models.

7. REFERENCES

[Cameron 1986]

Cameron, J.R. An overview of JSD. **IEEE Transactions on Software Engineering**, SE-12, 2 (February 1986), 222-240.

[Date 1982]

Date, C.J. An **Introduction to Database Systems**, 3rd edition, Addison-Wesley, 1982.

[Dolk 1986]

Dolk, D.R. Model management and structured modeling: The role of an information resource dictionary system. Submitted for publication.

[Dolk and Kirsch 1986]

Dolk, D.R. and Kirsch, R.A. A relational information resource dictionary system. Forthcoming in **Communications of the ACM**.

[Geoffrion 1985]

Geoffrion, A.M. **Structured Modeling**. Draft Research Monograph, UCLA Graduate School of Management, January 1985.

[Geoffrion 1986a]

Geoffrion, A.M. An introduction to structured modeling. Working Paper No. 338, Graduate School of Management, UCLA, June 1986.

[Geoffrion 1986b]

Geoffrion, A.M. Modeling approaches and systems related to structured modeling. Draft Working Paper No. 339, Graduate School of Management, UCLA, July 1986.

[Hoare 1978]

Hoare, C.A.R. Communicating sequential processes. **Communications of the ACM**, 21, 8 (December 1978), 666-677.

[Kottemann 1986]

Kottemann, J.E. Applying the Jackson system development and the Geoffrion structured modeling to systems description, Final Report for Government Contract #N62271-86-M-0209, September 1986.

[Kroenke 1983]

Kroenke, D. Database **Processing**, 2nd edition, Science Research Associates, 1983.

[Stone and Napoliello 1984]

Napoliello, M.F. and Stone, M.C. An examination of Jackson structured programming and design. Naval Postgraduate School Master's Thesis, June 1984.

[Stott 1986]

Stott, J.F. Representation and implementation of Jackson structured programs as structured models, Final Report for Government Contract #N62271-86-M-0209, October 1986.

[TRASANA 1978]

TRASANA Technical Memorandum 3-78 (ADB034635). Command, control, communications, and combat effectiveness model documentation, Vol. 1, Design Report, October 1978.

[Zeigler 1984]

Zeigler, B.P. **Multifaceted Modelling and Discrete Event Simulation**. New York: Academic Press, 1984.

APPENDIX A: FIPS IRDS ENTITY, ATTRIBUTE, AND RELATIONSHIP TYPES

DATA Entity Types

1. **DOCUMENT**: describes instances of human readable data such as tax forms or annual reports.

2. **FILE**: describes collections of records which represent an organization's data such as inventory files.

3. **RECORD**: describes instances of logically associated data such as a payroll record.

4. **ELEMENT**: describes an instance of data such as a social-security-number.

5. **BIT-STRING**: describes a string of binary digits.

6. **CHARACTER-STRING**: describes a string of characters.

7. **FIXED-POINT**: describes exact representations of numeric values.

8. **FLOAT**: describes exact representations of approximate numeric values.

PROCESS Entity Types

9. **SYSTEM**: describes a collection of processes and data such as an accounts-payable-system.

10. **PROGRAM**: describes a particular process such as print-accounts-payable-checks.

11. **MODULE**: describes a group of programs that are logically associated such as a sort-module.

EXTERNAL Entity Types

12. **USER**: describes an individual or organization that is using the IRDS such as the accounting-department

Table A-1: The Core System-Standard Schema Entity Types

Attribute Type	Entity Type USR	SYS	PGM	MDL	FIL	DOC	REC	ELE
ADDED-BY	S	S	S	S	S	S	S	S
(ALLOWABLE-RANGE)								P
ALLOWABLE-VALUE								P
CLASSIFICATION	P	P	P	P	P	P	P	P
CODE-LIST-LOCATION								P
COMMENTS	S	S	S	S	S	S	S	S
DATA-CLASS								S
DATE-ADDED	S	S	S	S	S	S	S	S
DESCRIPTION	S	S	S	S	S	S	S	S
DOCUMENT-CATEGORY						S		
DURATION-VALUE		S	S	S				
DURATION-TYPE		S	S	S				
(IDENTIFICATION-NAME) ALTERNATE-NAME	P	P	P	P	P	P	P	P
LAST-MODIFICATION-DATE	S	S	S	S	S	S	S	S
LAST-MODIFIED-BY	S	S	S	S	S	S	S	S
LOCATION	P	P	P	P	P	P		
NUMBER-OF-LINES-OF-CODE			S	S				
NUMBER-OF-MODIFICATIONS	S	S	S	S	S	S	S	S
NUMBER-OF-RECORDS					S			
RECORD-CATEGORY							S	
SECURITY	S	S	S	S	S	S	S	S
SYSTEM		S						

Table A-2: Core System-Standard Schema Attribute Types
(S=Single attribute; P=multiple attribute)

1. **CONTAINS**: describes a situation where an entity type con- other entity types (ex: Accounts-Payable-File CONTAINS Accounts-Payable-Record).

2. **PROCESSES**: describes a situation where an entity type acts upon another entity type (ex: Payroll-Program PROCESSES Payroll-Record).

3. **RESPONSIBLE-FOR**: describes an association between organizational entity type and other entity types indication organizational responsibility (ex: Accounting-Department RESPONSIBLE-FOR General-Ledger-File).

4. **RUNS**: describes an association between user and process entity types (ex: user RUNS program).

5. **GOES-TO**: describes a situation where one process transfers control to another process (ex: Accounts-Payable-Aging-Program GOES-TO Aging-Report-Program).

6. **DERIVED-FROM**: describes a situation where an entity is derived from another entity (ex: Annual-Report DERIVED-FROM Program-File).

7. **CALLS**: describes a situation where on entity calls another entity (ex: Data-Entry-Program CALLS Aging-Program).

8. **REPRESENTED-AS**: describes associations between ELEMENTs and certain data entities that document the ELEMENTs' format (ex: Employee-Name REPRESENTED-AS Character-String)

Table A-3: Core System-Standard Schema Relationship Types

(Note: The FIPS IRDS expresses relationships as ENTITYTYPE-RELSHIP-ENTITYTYPE, e.g. SYSTEM-CONTAINS-SYSTEM. We choose to represent relationships as **RELSHIP**(entitytype,entitytype) as below.)

CONTAINS(system,system)	PROCESSES(system,file)
CONTAINS(system,program)	PROCESSES(system,document)
CONTAINS(system,module)	PROCESSES(system,record)
CONTAINS(program,program)	PROCESSES(system,element)
CONTAINS(program,module)	PROCESSES(program,file)
CONTAINS(module,module)	PROCESSES(program,document)
CONTAINS(file,file)	PROCESSES(program,record)
CONTAINS(file,document)	PROCESSES(program,element)
CONTAINS(file,record)	PROCESSES(module,file)
CONTAINS(file,element)	PROCESSES(module,document)
CONTAINS(document,document)	PROCESSES(module,record)
CONTAINS(document,record)	PROCESSES(module,element)
CONTAINS(document,element)	PROCESSES(user,file)
CONTAINS(record,record)	PROCESSES(user,document)
CONTAINS(record,element)	PROCESSES(user,record)
CONTAINS(element,element)	PROCESSES(user,element)
RESP_FOR(user,file)	DERIVED_FROM(document,file)
RESP_FOR(user,document)	DERIVED_FROM(document, document)
RESP_FOR(user,record)	
RESP_FOR(user,element)	DERIVED_FROM(document,record)
RESP_FOR(user,system)	DERIVED_FROM(element,file)
RESP_FOR(user,program)	DERIVED_FROM(element,document)
RESP_FOR(user,module)	DERIVED_FROM(element,record)
	DERIVED_FROM(element,element)
RUNS(user,system)	DERIVED_FROM(file,document)
RUNS(user,program)	DERIVED_FROM(file,file)
RUNS(user,module)	DERIVED_FROM(record,document)
	DERIVED_FROM(record,file)
	DERIVED_FROM(record,record)
CALLS(program,program)	GOES_TO(system,system)
CALLS(program,module)	GOES_TO(program,program)
CALLS(module,module)	GOES_TO(module,module)

Table A-4: IRDS Relationships

APPENDIX B: BRIEF SUMMARY OF THE SQL DATABASE LANGUAGE

SQL is a query language for creating, modifying, and retrieving data residing in a relational database. The following is a very brief introduction to SQL. We cover only sufficient details to support the discussion in the report. A fuller treatment of SQL can be found in most database texts (e.g.: [Date 1982] or [Kroenke 1983]).

The following three relations describing a parts-supplier example are used for illustrative purposes:

SUPPLIER(**sid**, sname, status, city)

PARTS(**pid**, pname, color, weight)

SUP_PART(**sid**, **pid**, qty)

Relations are indicated in upper-case and attributes in lower-case. Key attributes are shown in bold-face. Each relation can be thought of as a file with each row of the relation corresponding to a record and each attribute corresponding to a field in the record.

Simple queries are expressed using the SELECT-FROM-WHERE syntax:

```
SELECT attribute(s)
FROM   relation(s)
WHERE  boolean_conditions
```

For example, the SQL equivalent of "list all supplier names from Dallas" is:

```
SELECT SNAME
FROM   SUPPLIER
WHERE  CITY = 'Dallas'
```

Several attributes can be specified in the SELECT clause:

```
SELECT SNAME, SID
FROM   SUPPLIER
WHERE  CITY = 'Dallas'
```

and Boolean conditions can be linked via AND or OR operators:

```
SELECT SNAME, SID
FROM   SUPPLIER
WHERE  CITY = 'Dallas' OR CITY = 'Atlanta'
```

More than one relation can be involved in a query as well. The following query lists all part names supplied by supplier Zukowski:

```
SELECT PNAME
FROM   SUPPLIER, PARTS, SUP_PART
WHERE  SNAME = 'Zukowski' AND SUPPLIER.SID = SUP_PART.SID AND
       SUPPLIER.PID = PARTS.PID
```

When there is uncertainty concerning which relation an attribute in the query belongs to, it's necessary to append the relation prefix to that attribute using the "." separator.

Subqueries can be used to formulate arbitrarily complex queries. The following lists supplier names for suppliers who supply part P2:

```
SELECT SNAME
FROM   SUPPLIER
WHERE  SID IN
      (SELECT SID
       FROM   SUP_PART
       WHERE  PID = 'P2')
```

The subquery (in parentheses) is executed first to identify the set of supplier id's who supply part P2. This requires accessing the SUP_PART relation. Once that set has been identified then the main part of the query is executed to find the corresponding supplier name for each supplier id in the set.

Views can be created which are equivalent to logical files. For example, we may want to create a view of red parts only:

```
CREATE VIEW REDPARTS AS
(SELECT PID, PNAME, WEIGHT
 FROM   PARTS
 WHERE  COLOR = 'red')
```

This view can be manipulated exactly like any other relation. The only difference is that physically this view will not be stored as a separate relation.

APPENDIX C: IRDS MACROS

Most RDBMS allow the user to define macros as a convenient way of invoking specific SQL queries. This frees the user from having to know the underlying syntax of the corresponding command and provides a concise means of executing complex commands. By defining an appropriate set of these macros, the database and model administrator can provide the user with a flexible toolkit for data and model management. This appendix lists a few such macros as they would appear in the ORACLE RDBMS environment. Please note in this context that the "&" has a special meaning in ORACLE unrelated to its use in structured modeling. Specifically, the "&" refers to an argument whose value must be prompted from the user when the macro is invoked.

IMPACT_OF_CHANGE

If we change a specified information resource (&ent_name, &ent_type), what other information resources will this have an impact upon?

```
SELECT ANAME,DNAME FROM ENTITY
WHERE ANAME IN
  (SELECT E1NAME FROM RELSHIP
   WHERE E2NAME='&ent_name' AND E2TYPE='&ent_type')  OR
      ANAME IN
  (SELECT E2NAME FROM RELSHIP
   WHERE E1NAME='&ent_name' AND E1TYPE='&ent_type');
```

REL_INTEGRITY

Display which entity-types can participate in which relationship-types. This is essentially a dump of Table A-4.

```
SELECT RNAME,E1NAME,E2NAME FROM RELSHIP
WHERE E1TYPE='ENT_TYPE' AND E2TYPE='ENT_TYPE';
```

Equivalent relationship-specific macros can be fashioned by substituting the relationship-type for RELSHIP as follows (e.g.: CONTAINS_INTEGRITY):

```
SELECT E1NAME, E2NAME FROM CONTAINS
WHERE E1TYPE='ENT_TYPE' AND E2TYPE='ENT_TYPE';
```

CHECK_INVALID

Identify any invalid relationship instances (i.e., instances which violate relationship-type integrity constraints) in the IRDS:

```
SELECT RNAME,E1NAME,E1TYPE,E2NAME,E2TYPE FROM RELSHIP
WHERE E1TYPE != 'ENT_TYPE' AND E2TYPE != 'ENT_TYPE' AND
      (E1TYPE,E2TYPE) NOT IN
        (SELECT E1NAME,E2NAME FROM RELSHIP
         WHERE E1TYPE = 'ENT_TYPE' AND E2TYPE='ENT_TYPE')
```

Again, this could be made relationship-specific by substituting for RELSHIP. All invalid instances could be deleted by replacing the SELECT ... FROM RELSHIP clause above with DELETE FROM RELSHIP.

CALL_SEQ

Determine proper calling sequences for structured modeling genus types.

```
SELECT E2NAME FROM CALLS
WHERE  E1NAME = '&genus_type' AND E2TYPE = 'ENT_TYPE'
```

GENUS_STRUCTURE

Determine whether the generic structure of a model violates any of the rules of structured modeling ("!" stands for a logical "NOT"):

```
SELECT E1NAME, E1TYPE, E2NAME, E2TYPE FROM CALLS
WHERE E1TYPE != 'ENT_TYPE' AND E2TYPE != 'ENT_TYPE' AND
      MODEL = '&model_name' AND
      (E1TYPE, E2TYPE) NOT IN
      (SELECT E1NAME, E2NAME FROM CALLS
       WHERE E1TYPE = 'ENT_TYPE' AND E2TYPE = 'ENT_TYPE')
```

MODULE_CHECK

Check that no genus belongs to more than one module as follows:

```
SELECT E1NAME, E2NAME FROM CONTAINS
WHERE  E1TYPE = 'MODULE' AND COUNT(*) > 1 AND
       MODEL = '&model_name' AND
       E2TYPE IN ('PE', 'CE', 'ATT', 'VA', 'TEST', 'FCN')
```

ADJACENCY

Determine the adjacency for a specific genus (i.e., its calling sequence) within a model:

```
SELECT E2NAME, E2TYPE
FROM CALLS WHERE E1NAME = '&genus' AND MODEL = '&model_name'
```

NATURAL_LANG

Display a natural language summary of a specific model for management purposes:

```
SELECT ENAME, ETYPE, INDEX, COMMENTS
FROM   GENUS
WHERE  MODEL = '&model_name'
```

MATH_SUMMARY

Display a mathematical summary of a model for modelers:

```
SELECT ENAME, ETYPE, INDEX, INDEX_STMT
FROM   GENUS
WHERE  MODEL = '&model_name'
```

APPENDIX D: LOGICAL STRUCTURE OF MODEL MANAGEMENT SYSTEM DATABASE

Table: ATT

CNAME	COLTYP	WIDTH	
ENAME	CHAR	15	(KEY)
DNAME	CHAR	30	
ADDED_BY	CHAR	15	
DATE_ADDED	DATE	7	
MOD_BY	CHAR	15	
DATE_MOD	DATE	7	
NMODS	NUMBER	0	
COMMENTS	CHAR	60	
SECURITY	CHAR	8	
IDXNAM	CHAR	4	
IDXSTMT	CHAR	30	
RANGE	CHAR	15	
RULE	CHAR	60	

Table: ATT_TYPE

CNAME	COLTYP	WIDTH	
ANAME	CHAR	10	(KEY)
DNAME	CHAR	30	
ADDED_BY	CHAR	15	
DATE_ADDED	DATE	7	
MOD_BY	CHAR	15	
DATE_MOD	DATE	7	
NMODS	NUMBER	5	
COMMENTS	CHAR	60	
SECURITY	CHAR	8	

Table: CALLS

CNAME	COLTYP	WIDTH	
E1NAME	CHAR	15	(KEY)
E1TYPE	CHAR	10	(KEY)
E2NAME	CHAR	15	
E2TYPE	CHAR	10	
ENAME	CHAR	15	

Table: CE

CNAME	COLTYP	WIDTH	
ENAME	CHAR	15	(KEY)
ETYPE	CHAR	10	(KEY)
DNAME	CHAR	30	
ADDED_BY	CHAR	15	
DATE_ADDED	DATE	7	
MOD_BY	CHAR	15	
DATE_MOD	DATE	7	
NMODS	NUMBER	0	
COMMENTS	CHAR	60	
SECURITY	CHAR	8	
IDXNAM	CHAR	4	
IDXSTMT	CHAR	30	
RANGE	CHAR	15	
RULE	CHAR	60	

Table: CONTAINZ

CNAME	COLTYP	WIDTH	
E1NAME	CHAR	15	(KEY)
E1TYPE	CHAR	10	(KEY)
E2NAME	CHAR	15	(KEY)
E2TYPE	CHAR	10	(KEY)

Table: ENTITY

CNAME	COLTYP	WIDTH	
ENAME	CHAR	15	(KEY)
ETYPE	CHAR	10	(KEY)
DNAME	CHAR	30	
ADDED_BY	CHAR	15	
DATE_ADDED	DATE	7	
MOD_BY	CHAR	15	
DATE_MOD	DATE	7	
NMODS	NUMBER	5	
COMMENTS	CHAR	60	
SECURITY	CHAR	8	
IDXNAM	CHAR	4	
IDXSTMT	CHAR	30	
RANGE	CHAR	15	
RULE	CHAR	60	

Table: ENT_TYPE

CNAME	COLTYP	WIDTH	
ANAME	CHAR	10	(KEY)
DNAME	CHAR	30	
ADDED_BY	CHAR	15	
DATE_ADDED	DATE	7	
MOD_BY	CHAR	15	
DATE_MOD	DATE	7	
NMODS	NUMBER	5	
COMMENTS	CHAR	60	
SECURITY	CHAR	8	

Table: FCN

CNAME	COLTYP	WIDTH	
ENAME	CHAR	15	(KEY)
ETYPE	CHAR	10	(KEY)
DNAME	CHAR	30	
ADDED_BY	CHAR	15	
DATE_ADDED	DATE	7	
MOD_BY	CHAR	15	
DATE_MOD	DATE	7	
NMODS	NUMBER	0	
COMMENTS	CHAR	60	
SECURITY	CHAR	8	
IDXNAM	CHAR	4	
IDXSTMT	CHAR	30	
RANGE	CHAR	15	
RULE	CHAR	60	

Table: GENUS

CNAME	COLTYP	WIDTH	
ENAME	CHAR	15	(KEY)
ETYPE	CHAR	10	(KEY)
DNAME	CHAR	30	
ADDED_BY	CHAR	15	
DATE_ADDED	DATE	7	
MOD_BY	CHAR	15	
DATE_MOD	DATE	7	
NMODS	NUMBER	0	
COMMENTS	CHAR	60	
SECURITY	CHAR	8	
IDXNAM	CHAR	4	
IDXSTMT	CHAR	30	
RANGE	CHAR	15	
RULE	CHAR	60	

Table: MODEL

CNAME	COLTYP	WIDTH	
ENAME	CHAR	15	(KEY)
ETYPE	CHAR	10	(KEY)
ADDED_BY	CHAR	15	
DATE_ADDED	DATE	7	
MOD_BY	CHAR	15	
DATE_MOD	DATE	7	
NMODS	NUMBER	0	
COMMENTS	CHAR	60	
SECURITY	CHAR	8	
IDXNAM	CHAR	4	
IDXSTMT	CHAR	30	
RANGE	CHAR	15	
RULE	CHAR	60	

Table: MODULE

CNAME	COLTYP	WIDTH	
ENAME	CHAR	15	(KEY)
ETYPE	CHAR	10	(KEY)
DNAME	CHAR	30	
ADDED_BY	CHAR	15	
DATE_ADDED	DATE	7	
MOD_BY	CHAR	15	
DATE_MOD	DATE	7	
NMODS	NUMBER	0	
COMMENTS	CHAR	60	
SECURITY	CHAR	8	

Table: PE

CNAME	COLTYP	WIDTH	
ENAME	CHAR	15	(KEY)
ETYPE	CHAR	10	(KEY)
DNAME	CHAR	30	
ADDED_BY	CHAR	15	
DATE_ADDED	DATE	7	
MOD_BY	CHAR	15	
DATE_MOD	DATE	7	
NMODS	NUMBER	0	
COMMENTS	CHAR	60	
SECURITY	CHAR	8	
IDXNAM	CHAR	4	
IDXSTMT	CHAR	30	
RANGE	CHAR	15	
RULE	CHAR	60	

Table: RELSHIP

CNAME	COLTYP	WIDTH	
RNAME	CHAR	15	(KEY)
E1NAME	CHAR	15	(KEY)
E1TYPE	CHAR	10	(KEY)
E2NAME	CHAR	15	(KEY)
E2TYPE	CHAR	10	(KEY)

Table: REL_TYPE

CNAME	COLTYP	WIDTH	
ANAME	CHAR	10	(KEY)
DNAME	CHAR	30	
ADDED_BY	CHAR	15	
DATE_ADDED	DATE	7	
MOD_BY	CHAR	15	
DATE_MOD	DATE	7	
NMODS	NUMBER	5	
COMMENTS	CHAR	60	
SECURITY	CHAR	8	

Table: TEST

CNAME	COLTYP	WIDTH	
ENAME	CHAR	15	(KEY)
ETYPE	CHAR	10	(KEY)
DNAME	CHAR	30	
ADDED_BY	CHAR	15	
DATE_ADDED	DATE	7	
MOD_BY	CHAR	15	
DATE_MOD	DATE	7	
NMODS	NUMBER	0	
COMMENTS	CHAR	60	
SECURITY	CHAR	8	
IDXNAM	CHAR	4	
IDXSTMT	CHAR	30	
RANGE	CHAR	15	
RULE	CHAR	60	

Table: VA

CNAME	COLTYP	WIDTH	
ENAME	CHAR	15	(KEY)
ETYPE	CHAR	10	(KEY)
DNAME	CHAR	30	
ADDED_BY	CHAR	15	
DATE_ADDED	DATE	7	
MOD_BY	CHAR	15	
DATE_MOD	DATE	7	
NMODS	NUMBER	0	
COMMENTS	CHAR	60	
SECURITY	CHAR	8	
IDXNAM	CHAR	4	
IDXSTMT	CHAR	30	
RANGE	CHAR	15	
RULE	CHAR	60	

"List all entity-types in the MMS."

```
UFI> SELECT * FROM ENT_TYPE;

ANAME      DNAME                          ADDED_BY        DATE_ADDE
---------- ------------------------------ --------------- ---------
MOD_BY          DATE_MOD       NMODS
--------------- --------- ----------
COMMENTS                                                    SECURITY
---------------------------------------------------------- --------

PE         PRIMITIVE_ENTITY               DOLK            21-FEB-86

Primitive entity in Structured Modeling

CE         COMPOUND_ENTITY                DOLK            21-FEB-86
Compound entity in Structured Modeling

ATT        ATTRIBUTE                      DOLK            21-FEB-86
Attribute in Structured Modeling

VA         VARIABLE_ATTRIBUTE             DOLK            21-FEB-86
Variable attribute in Structured Modeling

TEST       TEST_ELEMENT                   DOLK            21-FEB-86
Test element in Structured Modeling

FCN        FUNCTION_ELEMENT               DOLK            21-FEB-86
Function element in Structured Modeling

GENUS      GENUS                          DOLK            21-FEB-86
Genus may be PE, CE, ATT, VA, TEST, or FCN

MODEL      MODEL                          DOLK            21-FEB-86
A mathematical or software model

MODULE     MODULE                         DOLK            21-FEB-86
A programming or Structured Modeling module
```

"List all attribute-types in MMS."

```
UFI> SELECT * FROM ATT_TYPE;

no records selected
```

(Note: Attribute-types have not yet been entered.)

```
"List all relationship-types in the MMS."

UFI> SELECT * FROM REL_TYPE;

ANAME      DNAME                          ADDED_BY        DATE_ADDE
---------- ------------------------------ --------------- ---------
MOD_BY          DATE_MOD       NMODS
--------------- --------- ----------
COMMENTS                                                    SECURI
------------------------------------------------------------ ------

CALLS      ENTITY-CALLS-ENTITY            DOLK            21-FEB-86
Generic structure

CONTAINS   ENTITY-CONTAINS-ENTITY         DOLK            21-FEB-86
Modular structure


"List all entities currently entered in the MMS."

UFI> SELECT * FROM ENTITY;

ENAME           ETYPE      DNAME                          ADDED_BY
--------------- ---------- ------------------------------ ---------
DATE_ADDE MOD_BY          DATE_MOD       NMODS
--------- --------------- --------- ----------
COMMENTS                                                    SECURI
------------------------------------------------------------ ------
IDXSTMT                         RANGE
------------------------------- ---------------
RULE
------------------------------------------------------------

TRANSPORT       MODEL      TRANSPORTATION_MODEL           DOLK
21-FEB-86
Hitchcock-Koopmans transportation model


SOURCE          MODULE     SOURCE_DATA                    DOLK
21-FEB-86
Source (supply) data for transportation model


CUSTOMER        MODULE     CUSTOMER_DATA                  DOLK
21-FEB-86
Customer (demand) data for transportation model


DISTRIB         MODULE     DISTRIBUTION_DATA              DOLK
21-FEB-86
Distribution (transportation) data
```

```
PLANT           PE          SUPPLY_PLANTS                   DOLK
21-FEB-86
There is a list of plants


CUST            PE          CUSTOMER                        DOLK
21-FEB-86
There is a list of customers


LINK            CE          PLANT-CUSTOMER-LINK             DOLK
21-FEB-86
There are links from plants to customers
Select {PLANT}x{CUST}


SUP             ATT         PLANT_SUPPLY_CAPACITY           DOLK
21-FEB-86
Every plant has a supply capacity measured in tons
{PLANT}                         R+


DEM             ATT         CUSTOMER_DEMAND                 DOLK
21-FEB-86

Every customer has an exact demand measured in tons
{CUST}                          R+


COST            ATT         UNIT_TRANSPORTATION_COST        DOLK
21-FEB-86
Every link has a unit transportation cost ($/ton)
{LINK}                          R+


FLOW            VA          TRANSPORTATION_FLOW             DOLK
21-FEB-86
Every link may have a nonnegative flow (in tons)
{LINK}                          R+


T:SUP           TEST        SUPPLY_CONSTRAINT               DOLK

21-FEB-86
Is the total flow from a plant <= its supply capacity?
{PLANT}
SUMj(FLOWij) <= SUPi
```

```
T:DEM           TEST       DEMAND_CONSTRAINT              DOLK
21-FEB-86
Is the total flow to a customer = to its demand?
{CUST}
SUMi(FLOWij) = DEMj

TOTAL_COST      FCN        TOTAL_TRANSPORTATION_COST      DOLK
21-FEB-86
There is a total cost associated with all flows
1
SUMij(COSTij * FLOWij)


14 records selected.
```

"List all relationships entered in the MMS."

```
UFI> SELECT * FROM RELSHIP
```

RNAME	E1NAME	E1TYPE	E2NAME	E2TYPE
CALLS	CE	ENT_TYPE	PE	ENT_TYPE
CALLS	ATT	ENT_TYPE	PE	ENT_TYPE
CALLS	ATT	ENT_TYPE	CE	ENT_TYPE
CALLS	VA	ENT_TYPE	PE	ENT_TYPE
CALLS	VA	ENT_TYPE	CE	ENT_TYPE
CALLS	TEST	ENT_TYPE	ATT	ENT_TYPE
CALLS	FCN	ENT_TYPE	ATT	ENT_TYPE
CALLS	TEST	ENT_TYPE	VA	ENT_TYPE
CALLS	FCN	ENT_TYPE	VA	ENT_TYPE
CALLS	TEST	ENT_TYPE	TEST	ENT_TYPE
CALLS	TEST	ENT_TYPE	FCN	ENT_TYPE
CALLS	FCN	ENT_TYPE	FCN	ENT_TYPE
CALLS	FCN	ENT_TYPE	TEST	ENT_TYPE
CONTAINS	MODULE	ENT_TYPE	MODULE	ENT_TYPE
CONTAINS	MODULE	ENT_TYPE	CE	ENT_TYPE
CONTAINS	MODULE	ENT_TYPE	TEST	ENT_TYPE
CONTAINS	MODULE	ENT_TYPE	FCN	ENT_TYPE
CONTAINS	MODEL	ENT_TYPE	MODULE	ENT_TYPE
CALLS	T:DEM	TEST	FLOW	VA
CALLS	TOTAL_COST	FCN	FLOW	VA
CALLS	TOTAL_COST	FCN	COST	ATT
CALLS	SUP	ATT	PLANT	PE
CALLS	T:SUP	TEST	SUP	ATT
CALLS	DEM	ATT	CUST	PE
CALLS	T:DEM	TEST	DEM	ATT
CALLS	LINK	CE	PLANT	PE
CALLS	LINK	CE	CUST	PE
CALLS	COST	ATT	LINK	CE
CALLS	FLOW	VA	LINK	CE
CALLS	T:SUP	TEST	FLOW	VA
CONTAINS	TRANSPORT	MODEL	SOURCE	MODULE
CONTAINS	TRANSPORT	MODEL	CUSTOMER	MODULE
CONTAINS	TRANSPORT	MODEL	DISTRIB	MODULE

```
CONTAINS        TRANSPORT       MODEL       TOTAL_COST      FCN
CONTAINS        TRANSPORT       MODEL       T:SUP           TEST
CONTAINS        TRANSPORT       MODEL       T:DEM           TEST
CONTAINS        SOURCE          MODULE      PLANT           PE
CONTAINS        SOURCE          MODULE      SUP             ATT
CONTAINS        CUSTOMER        MODULE      CUST            PE
CONTAINS        CUSTOMER        MODULE      DEM             ATT
CONTAINS        DISTRIB         MODULE      LINK            CE
CONTAINS        DISTRIB         MODULE      COST            ATT
CONTAINS        DISTRIB         MODULE      FLOW            VA

44 records selected.
```

APPENDIX E: SOURCE CODE FOR MODEL MANAGEMENT SYSTEM

```
      PROGRAM PRISM
C=============================================================
C
C       Name: PRogram for Integrated Structured Modeling
C
C       Auth: Daniel R. Dolk
C             Code 54DK
C             Naval Postgraduate School
C             Monterey, CA  93943
C             408-646-2260
C             AV: 878-2260
C
C       Lang: Fortran 77
C
C       Comp: Vax 780
C
C       OS  : VMS
C
C=============================================================
C
      INCLUDE 'FLAGS.COM'

      CALL INTRO
C
C  ****  Accept query from user and parse it.
C
      QRY     = .TRUE.
100   CALL ACCQRY(QRY)
      CALL PARSE(QRY)
C
C  ****  Preprocess and execute command.
C
      CALL PROCES(QRY)
      IF (.NOT. EXIT) GO TO 100
      RETURN
C
      END
C
C
C
      SUBROUTINE INTRO
C
C=============================================================
C
C     Name: display INTROduction to program
C
C     Args: None
C
C     Rmks: Display logo, copyright, etc.
C
C=============================================================
```

```
C
      INCLUDE 'FILES.COM'
C
      CALL SKIPL(IU, 10)
      WRITE (IU,2000)
      RETURN
C
2000  FORMAT (' ',10X,'  ****************************************  ',/
     +        ' ',10X,' ****************************************** ',/
     +        ' ',10X,'**                                      **',/
     +        ' ',10X,'**              P R I S M               **',/
     +        ' ',10X,'**                                      **',/
     +        ' ',10X,'**            PRogram for               **',/
     +        ' ',10X,'**     Integrated Structured Modeling   **',/
     +        ' ',10X,'**                                      **',/
     +        ' ',10X,'**            Property of               **',/
     +        ' ',10X,'**     Naval Postgraduate School and    **',/
     +        ' ',10X,'**        U.S. Army TRASANA, 1986.      **',/
     +        ' ',10X,'**                                      **',/
     +        ' ',10X,' ****************************************** ',/
     +        ' ',10X,'  ****************************************  '
C
      END
C
C
C
      SUBROUTINE MODSTR(MODEL)
C
C=======================================================
C
C       Name: determine MODular STRucture of a model
C
C       Args: MODEL ---> Model name.
C
C       Rmks: Capture the modular structure for this particular
C             model.
C
C=======================================================
C
      INCLUDE 'ELEMS.COM'
      INCLUDE 'MODSTR.COM'
C
      CHARACTER*10 MODEL
      INTEGER*2 TOP
C
C  ****  We're trying to fill the CONTZ array by searching the
C   **   database for the MODEL name and any modules/genera which
C   **   are contained by the model.  This continues recursively
C   **   for all modules (note: genera are the leaves of the
C   **   modular tree and should not contain anything!) as we
C   **   work through the entire tree.
C
C  ****  Elements will be added as we go.  ELEM will be treated
C   **   as a last-in, last-out data structure with NELEM as the
C   **   current node in the tree and TOP as the total number of
```

```
C    **   (unique) nodes currently found.
C
       NCONT    = 0
       NELEM    = 1
       TOP      = 1
C
C   ****  The 1st element will always be the model.
C
       CALL SMOVE(ELEM, 1, MODEL, 1, 10)
C
C   ****  Now access the DB to find direct offspring of the current
C    **   element in the tree.
C
100    "SELECT E2NAME,E2TYPE FROM CONTAINZ WHERE E1NAME=ELEM(TOP) AN
        E1TYPE=TYPE(TOP);"
C
C   ****  If an offspring is found, move that element into the
C    **   element array and update the CONTZ array.
C
       IF (NULL) GO TO 200
         NELEM = NELEM + 1
         CALL SMOVE(ELEM(NELEM), 1, E2NAME, 1, 10)
         CALL SMOVE(TYPE(NELEM), 1, E2TYPE, 1, 10)
         NCONT = NCONT + 1
         CONTZ(1,NCONT) = NELEM
         CONTZ(2,NCONT) = TOP
         GO TO 100
C
C   ****  If an offspring not found, move current node to next
C    **   element in the ELEM array.
C
200    TOP      = TOP + 1
C
C   ****  If current element overflows the "top", we're done.
C
       IF (TOP .GT. NELEM) RETURN
C
C   ****  Only modules may have offspring.
C
         IF (ISCOMP(TYPE(TOP), 1, 'MDL ', 1, 4) .NE. 0) GO TO 200
           GO TO 100
C
       END
C
C
C
       SUBROUTINE GENSTR
C
C=============================================================
C
C        Name: determine GENeric STRucture
C
C        Args: None
C
C        Rmks: Generic structure is determined from the CALLS
```

```
C             relation in the DB.
C
C=============================================================
C
      INCLUDE 'FILES.COM'
      INCLUDE 'FLAGS.COM'
      INCLUDE 'ELEMS.COM'
      INCLUDE 'GENSTR.COM'
C
      INTEGER*2 FNDELM
      LOGICAL CYCLIC
C
      NCALLS  = 0
      DO 200 N=1,NELEMS
        IF (TYPE(N) .EQ. 'PE  ' .OR.
     +      TYPE(N) .EQ. 'MOD ' .OR.
     +      TYPE(N) .EQ. 'MDL ') GO TO 200
C
C  ****  Search DB for next instance of CALLS with this element
C   **   as the caller.
C
100       "SELECT E2NAME,E2TYPE FROM CALLS WHERE E1NAME=ELEM(N) AND
     +     E1TYPE=TYPE(N);"
          IF (NULL) GO TO 200
            NCALLS = NCALLS + 1
            IF (NCALLS .GT. 500) THEN
              CALL ERRMSG('Calling sequences too large for program_')
              FATAL = .TRUE.
              RETURN
            ENDIF
C
C  ****  Update CALLS array.
C
            CALLS(1,NCALLS) = N
            CALLS(2,NCALLS) = FNDELM(ELEM, TYPE, NELEM, E2NAME, E2TYPE
            IF (CALLS(2,NCALLS) .EQ. 0) THEN
              CALL ERRMSG('Undefined genus:_')
              WRITE (OU,2000) E2TYPE, E2NAME
              FATAL = .TRUE.
            ENDIF
            GO TO 100
200   CONTINUE
C
C  ****  Sort in ascending order by caller.
C
      CALL SRTVAR(CALLS, 2, NCALLS, 1, 1)
C
C  ****  Check for cyclicity.
C
      IF (NCALLS .GT. 0) FATAL = CYCLIC(CALLS, NCALLS, 0, 1, OU)
      RETURN
C
2000  FORMAT ('  ',A4,': ',A10)
C
      END
```

```
C
C
C
      LOGICAL FUNCTION CYCLIC(CALLS, NCALLS, IELEM, DISP, UNIT)
C
C=============================================================
C
C       Name: check for CYCLIC genus graph
C
C       Args: CALLS  ---> 2 row array where 1st row is caller and
C                         2nd row is callee.
C             NCALLS ---> No. of columns (calls) in CALLS.
C             IELEM  ---> If 0, check all elements for cyclicity;
C                         otherwise check only element IELEM.
C             DISP   ---> If 0, don't display cyclic elements;
C                         otherwise display on UNIT.
C             UNIT   ---> Unit for displaying cycles.
C
C=============================================================
C
      INCLUDE 'ELEMS.COM'
C
C  ****  STACK(1,i) = location in ELEM of element i;
C   **     "  (2,i) = location in CALLS of current occurrence of
C   **                this element.
C   **   MARK(i)    = 0 if i-th location in CALLS has not yet been
C   **                traversed; 1 otherwise.
C
      COMMON/LOCAL/ NSTACK, STACK(2,500), MARK(500)
      INTEGER*2 NSTACK, STACK, MARK
C
      INTEGER*2 CALLS(2,NCALLS), IELEM, DISP, UNIT
C
      DO 100 M=1,500
        MARK(M) = 0
100   CONTINUE
C
      CYCLIC  = .FALSE.
      DO 700 N=1,NCALLS
        IF (MARK(N) .NE. 0) GO TO 700
C
C  ****  Set up 1st 2 elements of STACK.
C
         MARK(N) = 1
         NSTACK  = 2
         STACK(1,1) = CALLS(1,N)
         STACK(2,1) = N
         STACK(1,2) = CALLS(2,N)
200      CALL BINSRC(CALLS, 2, NCALLS, STACK(1,NSTACK), 1, 1,
     +                STACK(2,NSTACK))
         IF (STACK(2,NSTACK) .LE. 0) GO TO 400
C
C  ****  Mark this entry in CALLS so we don't reevaluate.
C  ****  Continue adding to stack.
C
```

```
300         MARK(STACK(2,NSTACK)) = 1
            NSTACK = NSTACK + 1
            STACK(1,NSTACK) = CALLS(1,STACK(2,NSTACK-1))
            GO TO 200
C
400       NSTACK = NSTACK - 1
          IF (NSTACK .LE. 1) GO TO 700
            DO 500 I=2,NSTACK
              I1    = I - 1
              DO 500 J=1,I1
                IF (STACK(J) .EQ. STACK(I)) THEN
                  CYCLIC = .TRUE.
                  IF (DISP .GT. 0) CALL DSPSTK(UNIT, STACK, NSTACK,
     +                                         ELEM, TYPE)
                  GO TO 600
                ENDIF
500         CONTINUE
C
600         ICALL = STACK(2,NSTACK)
            IF (CALLS(1,ICALL) .EQ. CALLS(1,ICALL+1)) THEN
               STACK(2,NSTACK) = ICALL + 1
               GO TO 300
            ELSE
               NSTACK = NSTACK - 1
               IF (NSTACK .GT. 1) GO TO 600
            ENDIF
700   CONTINUE
      RETURN
C
      END
```

DISTRIBUTION LIST

	Number of Copies
Professor Daniel R. Dolk Code 54DK Naval Postgraduate School Monterey, CA 93943	5
Professor Arthur M. Geoffrion Graduate School of Management UCLA 405 Hilgard Avenue Los Angeles, CA 90024	2
Professor Jeffrey Kottemann University of Hawaii College of Business Administration 2404 Maile Way Honolulu, HI 96822	1
Colonel Charles Macchiaroli TREM Naval Postgraduate School Monterey, CA 93943	2
Mr. Ben Morgan U.S. Army TRADOC Systems Analysis Activity Attn: ATOR-TCA White Sands Missile Range, NM 88002-5502	10
Professor Michael Sovereign Code 74 Naval Postgraduate School Monterey, CA 93943	1
Professor Jack Stott University of Hawaii College of Business Administration 2404 Maile Way Honolulu, HI 96822	1
Administrative Sciences Department Code 54 Naval Postgraduate School Monterey, CA 93943	1
Computer Center Library Code 0141 Naval Postgraduate School Monterey, CA 93943	1

Defense Technical Information Center 2
Cameron Station
Alexandria, VA 23314

Knox Library 2
Code 0142
Naval Postgraduate School
Monterey, CA 93943

Office of Research Administration 1
Code 012
Naval Postgraduate School
Monterey, CA 93943